An Empirical Study of Container Image Configurations and Their Impact on Start Times

This is the author version of the work. Original publication available soon in IEEE Xplore

Martin Straesser*, André Bauer†, Robert Leppich*, Nikolas Herbst*, Kyle Chard†, Ian Foster†, Samuel Kounev*
*University of Würzburg, Würzburg, Germany
Email: {martin.straesser, robert.leppich, nikolas.herbst, samuel.kounev}@uni-wuerzburg.de
†University of Chicago, Chicago, IL, USA
Email: {andrebauer, chard}@uchicago.edu, foster@cs.uchicago.edu

***Abstract*—A core selling point of application containers is their fast start times compared to other virtualization approaches like virtual machines. Predictable and fast container start times are crucial for improving and guaranteeing the performance of containerized cloud, serverless, and edge applications. While previous work has investigated container starts, there remains a lack of understanding of how start times may vary across container configurations. We address this shortcoming by presenting and analyzing a dataset of approximately 200,000 open-source Docker Hub images featuring different image configurations (e.g., image size and exposed ports). Leveraging this dataset, we investigate the start times of containers in two environments and identify the most influential features. Our experiments show that container start times can vary between hundreds of milliseconds and tens of seconds in the same environment. Moreover, we conclude that no single dominant configuration feature determines a container's start time, and hardware and software parameters must be considered together for an accurate assessment.**

***Index Terms*—container, start time, docker, empirical study**

I. Introduction

Containers have become the predominant deployment method for modern cloud applications and a key enabler of the wide adoption of microservice architectures [1], [2]. They are now used in all areas of distributed computing, from traditional cloud to edge and serverless computing. A leading solution for containerization is Docker containers; the popular image repository Docker Hub, with over nine million accounts, has around 42 billion image downloads every quarter [3]. The container ecosystem gets even bigger if we include other registries like GitHub Packages[1] or other container management frameworks like Podman.[2]

One regularly mentioned advantage of containers over other virtualization technologies like virtual machines, and an important driver of adoption, is their faster start times. Understanding start times is essential for many application areas. For example, the fact that containers permit start times of a few seconds enabled the broad usage of serverless computing. However, start times remain an active research field in this domain as they are critical factors for desired rapid scale-ups [4], [5]. Likewise, the optimization of container deployment processes also plays an important role in edge computing [6], [7]. Furthermore, accurate knowledge of start times can also be helpful when planning autoscaling and resilience of microservice applications [8]. While the fact that containers have faster start times than virtual machines is well-established in the scientific literature [9]–[11], relatively little is known about variations in start times between different containers. Understanding the factors that influence container start times may help developers build containers with fast start times and define rules and best practices for creating such lightweight containers. In summary, start times are a critical performance factor for cloud, serverless, and edge applications. No prior work has examined start time variations across a wide variety of containers to make statements about influencing factors based on a large number of measurements.

We present here an analysis of how various container characteristics (e.g., image size, number of volumes) influence a container's start time. In this work, we consider the start time as a part of the container's readiness time. The latter includes the image download time and the application-specific setup time in addition. In contrast to the readiness time, the start time can be evaluated without knowledge about the container's file system and start command and independently of network parameters. For this analysis, we use about 200,000 open-source images retrieved from Docker Hub by a web crawler. This dataset is a mix of the most popular and recent Linux amd64 images queried in April 2022. We determine various properties of these images, determine the empirical distributions of those properties, and analyze their variance using principal component analysis. We then measure start times for a subset of these images in two test environments: a public cloud and a local testbed. We apply regression analysis techniques to determine the relevance of each image configuration parameter in predicting start time. We find that start times can vary between hundreds of milliseconds and tens of seconds in the same environment. However, we also show that these variations cannot be explained by looking at single configuration parameters and that hardware and software parameters must be considered together for an accurate assessment.

The goal of this paper is to raise awareness among researchers and operators of the significant differences in container start times and to provide a starting point for future

[1]https://github.com/features/packages
[2]https://podman.io/

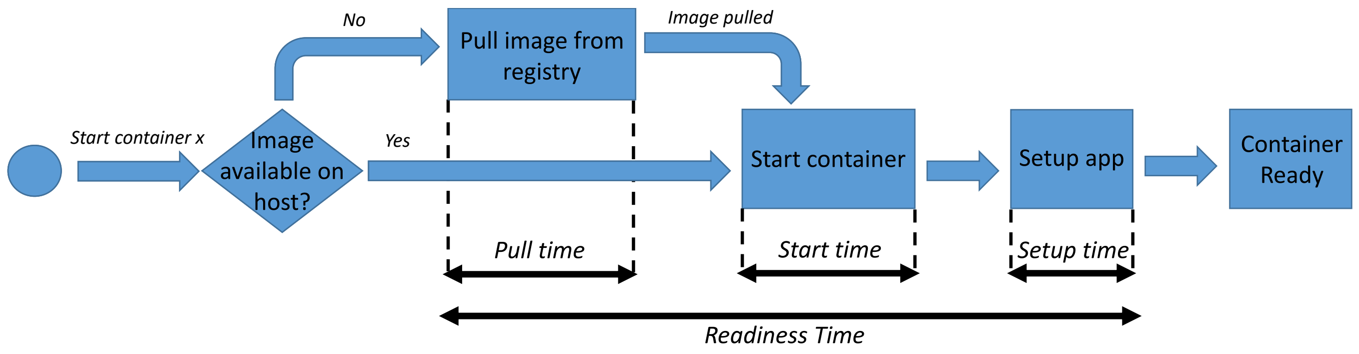

Fig. 1. Container readiness process and performance metrics.

investigations and predictions of start times. In summary, this paper makes two primary contributions:

- We present a large dataset[3] of open-source Docker Hub images and characterize the dataset;
- We analyze influencing factors for start times with measurements of more than 1000 container images taken from two different computing environments.

The remainder of this paper is structured as follows: In Section II, we provide background information on container start times and clarify the scope of this paper. Section III focuses on our image dataset, its characteristics, and how it has been acquired. Section IV features our methodology for the start time analysis, while Section V presents our results. In Section VI, we discuss the overall findings and limitations of our work. Section VII identifies related work and the differences to this paper, while Section VIII draws conclusions.

II. Foundations

This section presents the foundations of this work and defines our scope. Since we analyze start times based on image configurations, it is essential to understand container images and their components. The standard format for container images is the image format specification of the Open Container Initiative (OCI) [12]. According to this specification, a container image consists of a manifest, the file system layers, and a configuration. In this work, the configuration plays a major role, which contains information about exposed ports, volumes, the start command, and more. The default configuration of an image is the configuration that the creator of the image has set. A more detailed discussion of the relevant configuration entries is given in Section III.

Container managers are responsible for managing, modifying, and starting these OCI images. We focus on Docker as it is the most widely used container manager [13]. Docker can display the image configuration by the *inspect* command. The container manager uses an OCI runtime [14] to execute containers. The container runtime interacts with the operating system and uses control groups and namespaces to build an isolated execution environment for the container. We focus here on Docker's default runtime, *runc*. Section VII discusses related work, including with other container managers and runtimes.

In the following, we focus on container start times. The start time is a part of the container readiness time, as shown in Figure 1. The readiness time is the time taken from when the container is transferred from a registry until it can execute production workload; for web applications, this usually means processing user requests. This state of readiness is checked on container orchestration platforms like Kubernetes by so-called readiness probes. The total readiness time consists of three main components. *Pull time*, the time required to transfer the image from a registry to the host, depends mainly on network parameters. Note that no image transfer is necessary if the required image is already present on the host. *Start time*, the focus of this paper, is the time required to create a running container from an image. The user usually sends the start command to the container manager, which checks all requirements for the container to start and passes the information to the container runtime. The start process is further specified in the OCI runtime specification [14]. Essentially, a namespace is created, and the root file system for the container is provided. Potential factors influencing start time are, on the one hand, the underlying hardware and software stack and, on the other hand, image configuration parameters. We focus here on the latter. With our large dataset, we can test many different images and thus make statements about image parameters. Such a comprehensive test is not possible with the hardware and software parameters due to the vast number of possible variants. We use two test environments: Google Cloud VMs and bare-metal servers.

The start time is the timespan from receiving the start command to when the container starts running its entrypoint executable. This time can be evaluated without knowing the container's content and function, and indeed can be determined even for containers that cannot actually perform their intended function. Consider an example container that tries to connect to a database at an unreachable IP address. This container would never reach the ready state and would, instead, probably crash after a certain timeout. However, its start time can still be determined because the start process ends immediately

[3]https://doi.org/10.5281/zenodo.7602500

TABLE I
OVERVIEW OF FEATURES IN THE EXTRACTED DATASET.

Category	Feature	Type	Description
Metadata	meta_repo_digest	String	A SHA-256 hash which is used to uniquely identify and download the image from Docker Hub
	meta_architecture	String	The CPU architecture which the binaries in the image are built to run on
	meta_os	String	The name of the operating system which the image is built to run on
	meta_docker_version	String	The Docker version used to built this image
I/O Streams	io_attach_stdin	Boolean	Determines whether the console should be attached to the container process stdin stream
	io_attach_stdout	Boolean	Determines whether the console should be attached to the container process stdout stream
	io_attach_stderr	Boolean	Determines whether the console should be attached to the container process stderr stream
	io_tty	Boolean	Determines whether the console should pretend to be a TTY when attached
	io_open_std_in	Boolean	Determines whether the container process stdin stream should be kept open even if console not attached
	io_std_in_once	Boolean	Determines whether the container process received input from stdin stream at least once
Start Command	cmd_args	Integer	Length of list of arguments to use as the command to execute when the container starts
	cmd_envvars	Integer	Number of environment variables set per default when the container starts
	cmd_additional_args	Integer	Length of list for additional arguments to the containers entrypoint
File System	fs_volumes	Integer	Number of volumes to create/use by default
	fs_size	Integer	Size of the image in bytes
	fs_virtual_size	Integer	Virtual size of the image in bytes
	fs_graph_driver_name	String	Name of the image's graph driver
	fs_root_fs_type	String	Name of the file system type used in the image
	fs_layers	Integer	Number of root file system layers
Networking	net_ports	Integer	Number of ports to expose by default

when the entrypoint command is executed. Therefore, the only condition for examining the start time is that the container is startable. We discuss exceptional cases for when a container is not startable in Section III.

The start time is followed by the *setup time*, i.e., the time taken for a started container to reach the ready state. This timespan depends mainly on the entrypoint command and the actual use case of the container. In the example of the preceding paragraph, a connection to a database must be established. In many application areas (e.g., serverless computing), a programming language runtime (e.g., Python) has to be initialized. Analysis of setup time, and thus the total readiness time, requires knowledge of container internals and external dependencies. Consequently, a generalization of start times to readiness times and a broad-based automated analysis of those is not possible.

III. DATASET

We now present the dataset used in this paper, describing first how we acquired the data and then various characteristics of the data. The dataset and further information are included in our replication package.[4]

A. Dataset acquisition

To investigate variation in image configuration parameters and the influence of those parameters on start times, we first created a dataset containing a broad selection of container images. We choose Docker Hub as a widely used container image repository [13] and use a web crawler (based on the Docker Hub Explore function[5]) to extract a large number of image names from this repository. Specifically, we first did a substring search on the Docker Hub "most popular" and "recently updated" categories with search strings up to a length of three characters with all letter combinations in the range a–z (i.e., from `a` to `zzz`). Then, we included the images displayed on the start page. In this way, we obtained a selection of both the most popular images (the most downloaded) and an unbiased set of less popular images. We included only Linux amd64 architecture images in our dataset to ensure compatibility with our test machines. Both the Linux operating system and amd64 architecture are widely used in modern public clouds. We queried the image data in April 2022.

After removing duplicates, we were left with 286,294 image names. The number of images we could crawl is limited by the fact that the Docker Hub Explore function limits the number of results per search request to 500. Next, we attempted to download the associated image for each of these names, capture its configuration data (including unique identifier hash), and run it once. We eliminated from further consideration images that could not be downloaded (e.g., because further authentication was required) or that were not runnable (e.g., because a lack of a default start command or an invalid root file system meant that the start command is not executable). A more detailed description of the error types we encountered can be found in our replication package[4]. These filtering steps left us with 200,986 valid and pullable images.

B. Dataset characterization

We characterize each image in the dataset with 20 features extracted from its default configuration. We downloaded all images and analyzed their OCI configuration properties to decide which features to include for our analysis. The OCI image configuration specification [12] defines required and optional configuration properties. The 20 resulting dataset features include required properties (e.g., *fs_root_fs_type*) and optional properties (e.g., *net_ports*) where a value has been

[4]https://doi.org/10.24433/CO.4595026.v2
[5]https://hub.docker.com/explore

TABLE II
FREQUENCIES OF DIFFERENT DOCKER VERSIONS IN DATASET.

Version (M/Y)	# Images	Version (M/Y)	# Images
20.10 (12/2020)	19,647	1.7.x (06/2015)	78
19.03 (07/2019)	42,479	1.6.x (04/2015)	158
18.x (01/2018)	53,697	1.5.x (02/2015)	51
17.x (03/2017)	34,886	1.4.x (12/2014)	18
1.13.x (01/2017)	4821	1.3.x (10/2014)	31
1.12.x (07/2016)	14,089	1.2.x (08/2014)	11
1.11.x (04/2016)	3440	1.1.x (07/2014)	10
1.10.x (02/2016)	1196	1.0.x (06/2014)	19
1.9.x (11/2015)	340	0.x (03/2013)	7
1.8.x (08/2015)	205	N/A	25,803

set for all 200,986 images. Note that this study does not consider other optional properties set for only a few images. We provide a complete list and description of the extracted features in Table I and our replication package[4]. The features' meaning was taken from the OCI image specification and the Docker documentation [15]. We consider five feature categories: metadata, I/O streams, networking, file system, and information about the start command. Selected features are further described in Section V if they were found to affect start time.

In the following, we present more detail about selected features and their distributions. First, we focus on the Docker version, as it indicates when the images were built/published most likely. Table II shows the number of times different Docker versions appear in our dataset. We also labeled each version with its release date.[6] Images for which no Docker version is specified or cannot be unambiguously assigned are marked as N/A. We see that while most images are likely from the last five years, some older images are also included in the dataset.

Figure 2 shows, with a logarithmic scale, the distributions of 10 other features in the dataset. The first four features in the upper row are the boolean settings mentioned in Table I, considering the I/O settings of the container. We see that these features are all similarly distributed and that, in most cases, no streams are attached by default. In the upper right corner, we see the exposed ports as a networking feature. As expected, many containers expose only a few ports by default. However, the dataset also contains two containers that expose the maximum number of 65,536 ports. In the bottom row of Figure 2, we see on the left side the number of environment variables and the size of the list of arguments to use as the command to execute when the container starts. Like the exposed ports, we see the highest frequency at the value of zero but again with a relatively large value range in total.

The three features shown in the bottom right give quantitative information about the container file system. For the number of volumes (fs_volumes), we see a distribution with a maximum at zero and a sharply decreasing frequency. However, the dataset also contains two outliers with 21 and 32 volumes. We observe more complex distributions regarding the number of file system layers (fs_layers) and the image size (fs_size). The containers in our dataset have a minimum of 1, an average of 10.84, a median of 9, and a maximum of 125 root file system layers. The observed maximum is equal to the technical maximum [16], which means that our dataset represents the largest range possible. The image sizes show an even higher scatter. The minimum image size in our dataset is 45 bytes, the mean is 837.3 MB, and the median is 398.6 MB. The largest image has a size of about 90.4 GB.

In summary, the dataset has a good diversity across many features. In the following, we examine the total variance of the dataset in more detail using Principal Component Analysis (PCA) [17]. PCA can be used to reduce the number of features in a dataset, for example, to apply machine learning algorithms more efficiently. In particular, constant and (nearly) linearly dependent features are identified. PCA calculates the eigenvectors of the covariance matrix, the so-called principal components. The principal components are linear combinations of the original features. By looking at the proportion of the variance of each principal component in the sum of the variances of all principal components (often referred to as *explained variance ratio*), one gets an idea of how many principal components are needed to represent a certain share of the total dataset variance. If only a few components are needed to capture a large proportion of the variance, the dataset contains many linearly dependent or constant features.

Figure 3 shows a cumulative view of the explained variance ratio. As preprocessing steps, we removed the metadata features and scaled the remaining data using z-score normalization. We see that the first principal component covers about 31% of the total variance and that, with ten components, about 99.9% of the total variance can be described. These numbers show that our dataset contains about six features with a negligible variance or that are linearly dependent on other features. All other features make some non-negligible contribution to the total variance, and their influence on the start time can be investigated meaningfully.

IV. METHODOLOGY

This section explains our methodology for using data from the presented dataset to determine the factors influencing container start times.

A. Study overview

As explained in Section III, we analyzed 200,986 distinct images in this study. Our goal is to measure the container start time and then investigate the impact of the image parameters. To this end, we studied how start times vary for different containers as well as for repeated starts of a given container image. We assume that besides the image parameters, the hardware and software of the deployment environment play a role. To account for this, we measure the start times in two test environments: on bare-metal servers in a self-hosted cloud and on Google Cloud VM instances. Time, cost, and technical limitations prevented us from testing all 200,986 images multiple times. The main limitations are image download limits

[6]https://docs.docker.com/engine/release-notes/

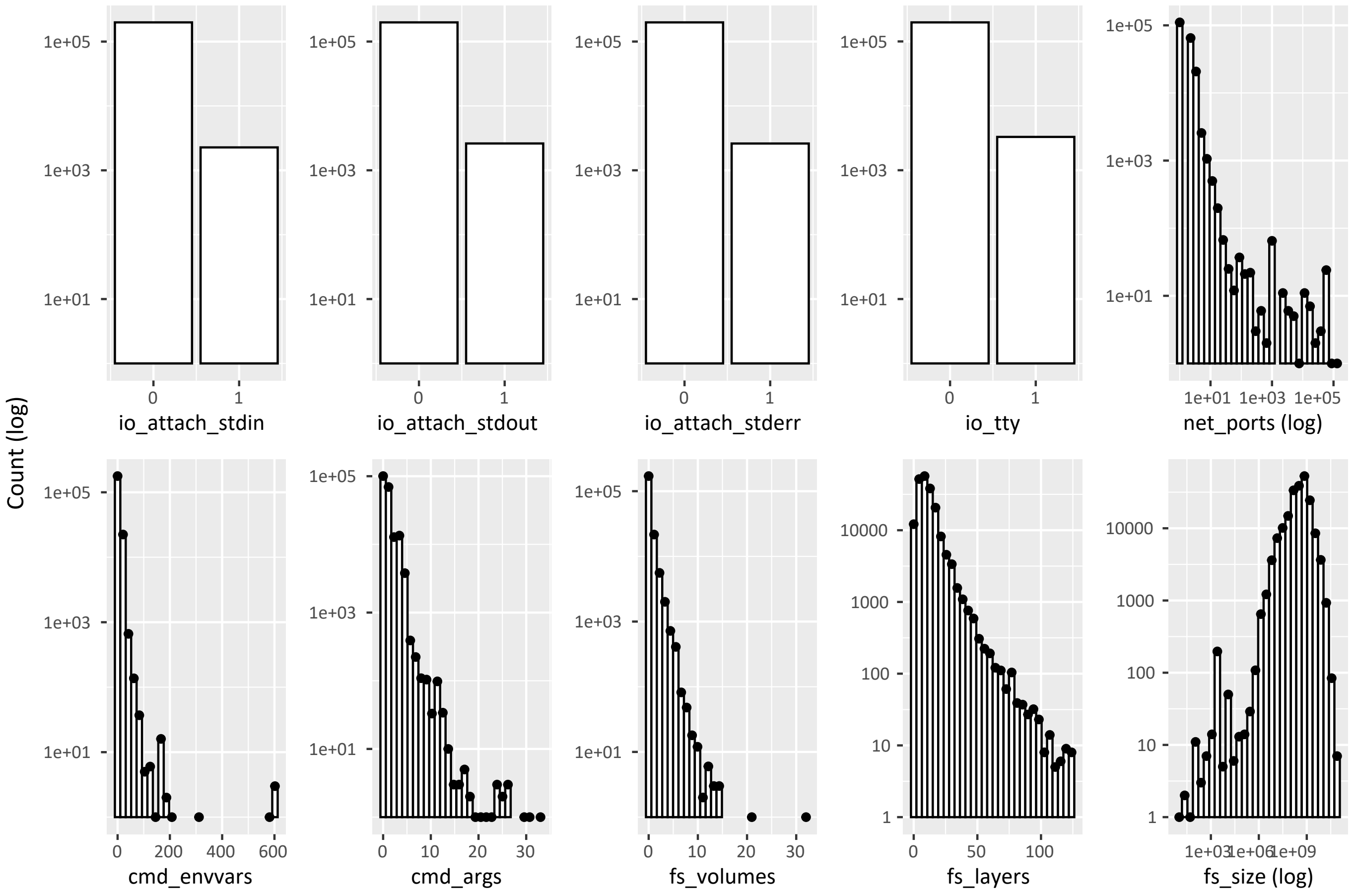

Fig. 2. Empirical distributions of selected features.

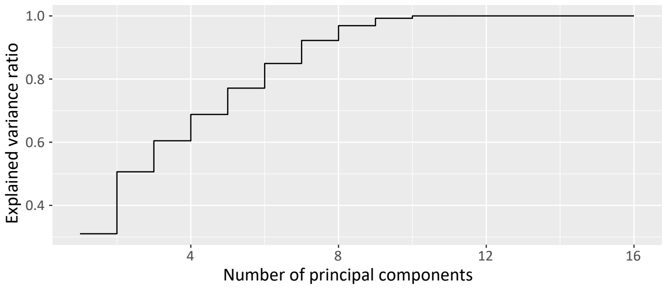

Fig. 3. Cumulative explained variance ratio by principal components.

enforced by Docker Hub and the induced costs for the virtual machines. Consequently, we tested a sample of images from the entire dataset. In the following sections, we describe our sampling and measurement processes.

B. Sample selection

There are different sampling techniques that can be applied to draw a sample from our dataset. We aim to select a sample that, on the one hand, reflects well the typical configurations (i.e., the most frequent image parameter combinations) but, on the other hand, also takes outliers into account (e.g., images of enormous size). As shown in Section III, our dataset contains many image and parameter combinations. To cope with this heterogeneity, we applied stratified sampling [18] instead of simple random sampling because the latter may fail to capture outliers and extreme values properly.

Figure 4 shows our sampling process. In stratified sampling, data is divided into so-called strata. The idea is to divide a dataset into subpopulations that are as homogeneous as possible. Random sampling is then used to draw a sample from each stratum. The combined strata samples form the total dataset sample. Accordingly, the first step in our sampling process is to divide the dataset into strata. For this, we evaluate different clustering algorithms and perform a hyperparameter study. An overview of the evaluated algorithms and parameter combinations is given in our replication package[4]. The clustering algorithms can be applied to the dataset directly; the only feature modified for clustering is the Docker version number which we reduced to the major version and then applied one-hot encoding. The best clustering algorithm in our hyperparameter study was k-means clustering with 13 resulting strata. We used the silhouette coefficient [19] to evaluate the resulting clusters, obtaining a value of 0.93 for the final clustering. After clustering, we used a decision tree

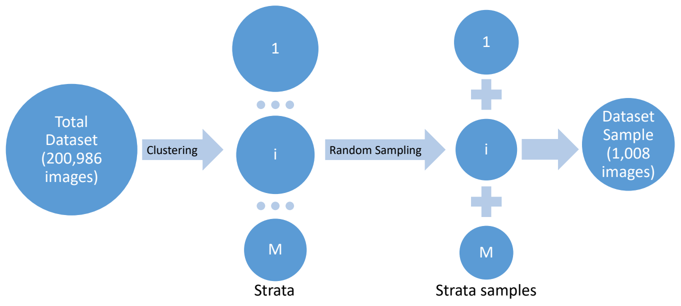

Fig. 4. Sampling process.

to capture the feature importance of the individual image parameters for the clustering. As expected, the image size is the most important parameter in the clustering (because it has the highest value range), followed by the number of file system layers. A quantitative breakdown of the feature importance can be found in our replication package[4]. The largest stratum contains 52,574 images, while the smallest contains 351 images.

In the next step, we need to determine how many images to take from each stratum. For this, we use disproportionate allocation [18], meaning that the stratum's size and variance play a role in deciding how many images to take. The number of images n_i taken from a stratum i is given by

$$n_i = \left\lceil N \cdot \frac{d_i/D \cdot s_i}{\sum_{j=0}^{M-1} d_j/D \cdot s_j} \right\rceil, \quad (1)$$

where N is the target sample size from the dataset, M is the number of strata, d_i is the number of images in stratum i, D is the total number of images, and s_i is the standard deviation in stratum i; we use the square root of the total variance to measure the standard deviation of a stratum. The total variance is a generalization of the variance for multidimensional datasets and is defined as the trace of the covariance matrix [20]. We set the target sample size N to 1000; the number of strata M is 13. As a result, the number of images selected from each stratum varies between 3 and 238, and the total number of images in the dataset sample is 1008; the ceil operation explains the difference of eight to the parameter N in the formula. In summary, through our sampling process, we reflect the diversity of the entire dataset in our sample. Visualizations of the resulting sample composition can be found in our replication package[4].

C. Test environments

We use two test environments in this paper. The first environment consists of three bare-metal servers with identical hardware. The servers are HP ProLiant DL360 Gen9 with Intel(R) Xeon(R) CPU E5-2640, 2x16 GiB HP 752369-081 DDR4 RAM, and a 500 GB HDD disk of type HP MB0500GCEHE. The second test environment consists of three Google Cloud VMs of type e2-medium running in the us-central1-c zone with a 100 GB zonal balanced persistent disk. All test machines use Ubuntu 22.04 LTS as the operating system and Docker v20.10.21, containerd v1.6.10, and runc v1.1.4.

D. Measurement process

In addition to the three test machines, we deployed a master VM in each environment to host the image and result databases. The master VM sends jobs, each defined by a specific image hash and a desired number of repetitions, to the test machine that is to execute them. The execution of a job consists of three steps. First, the required image is downloaded from Docker Hub using the *docker pull* command. Then, the container is started with its default configuration using the *docker run* command. We extract the time when the Docker daemon received the run command from its logs and the time when the container finished the start process from the output of the *docker inspect* command. The start time is the difference between these two timestamps. Note that we only use data that Docker collects and do not manually generate any timestamps. The pull time is not recorded and does not play a role in the tests. Once the timestamps have been recorded, and the job has finished, the container and image are removed from the test machine.

For the 1008 sampled images from Section IV-B, we run 30 repetitions for each image, resulting in 30,240 container starts per environment. We use the randomized multiple interleaved trials methodology [21] to account for the variability of the cloud environment and reduce caching effects. This results in the order of repetitions being random. Moreover, the master VM randomly distributes jobs to test machines. Besides the Docker daemon, only a tiny web service runs during the measurements on the test machines. It communicates with the master VM and is executed as an ordinary process, not in a container.

V. Results

We use a three-step zoom-out approach for the description of our results. Each step is associated with a research question. In the first step, we investigate how the start time of one particular container varies between repeated starts. In this step, we assess the start time variance for one fixed feature combination in our dataset. In the second step, we analyze the impact of the image configuration on the start time. Here, we consider the measurements from the Google Cloud test environment. This means we look at the start times of different image configurations in one environment. The last step deals with the generalizability of the results. We investigate whether the results from the Google Cloud environment are transferable to our self-hosted environment. Here, our focus lies on the impact of the hosts, including their hardware and software. In other words, we evaluate the start times of different image configurations in different environments.

Before we present a detailed analysis of the start times, it is worth looking at the overall statistics to understand the magnitude of the start times. In the Google Cloud environment, the minimum start time is 277 ms, the mean is 1886 ms, the

TABLE III
COV FOR START TIMES OF ONE IMAGE CONFIGURATION.

Environment	Coefficient of Variation				
	Min	Median	Mean	95th Percentile	Max
Google Cloud	0.023	0.153	0.179	0.389	0.966
Self-hosted	0.051	0.177	0.270	0.788	3.069

median is 1689 ms, and the maximum is 17,605 ms. In the self-hosted cloud, the minimum is 1241 ms, the mean is 8417 ms, the median is 6470 ms, and the maximum is 426,687 ms.

A. What is the start time variation for one image?

As described in Section IV, we start each image configuration 30 times. We consider the coefficient of variation (CoV) as a metric to quantify the variability of the start time. Consequently, we obtain a CoV for each of our 1008 sampled images. Table III shows the minimum, mean, median, 95th percentile, and maximum CoV for the two test environments.

We see that the self-hosted environment exhibits a larger variation than the Google Cloud, as evidenced by all statistical values, although the median for both environments is about the same at 15.3% and 17.7%, respectively. The cause of the higher variation in the self-hosted environment can be explained by the storage technology used. While SSDs back the Google Cloud VMs, the self-hosted servers have conventional HDDs. We conducted I/O benchmarking runs [22] that indicate that the CoV of submission and completion latency in read-intensive scenarios is about 2-11 times higher in the self-hosted environment than in the Google Cloud.

In our analysis, we had a particular focus on the outliers that have a high CoV. The 95th percentile shows that there are only a few outliers in both environments. We analyzed the 5% images with the highest CoV from both environments, in total 51 per environment. Only four images were outliers in both test environments. We further analyzed the configurations of these images but found no clear relationship to any single dataset feature. This is also in line with the results of the next section.

B. How do image configuration parameters impact start time?

In the following, we perform a regression analysis to determine factors that influence the container start time. First, we focus on the results in the Google Cloud environment, as this environment is representative and widely used in practice. The results of our second test environment are then discussed in Section V-C. Out of our initial 20 features, we eliminated the six textual features from Table I along with two features that are linearly dependent on other features (*fs_virtual_size* and *io_attach_stderr*). Another set of three features has constant values for all sample elements (*io_std_in_once*, *io_open_std_in*, *cmd_additional_args*). From the definition of these features (see Table I), it is clear that their values are expected to be zero/false by default and only change if the container starts in a non-default configuration. After these exclusion steps, we have a set of nine features for further evaluation.

TABLE IV
UNIVARIATE AND MULTIVARIATE LINEAR REGRESSION COEFFICIENTS AND p-VALUES.

Feature	Univariate Regression		Multivariate Regression	
	β_1	p	β_1	p
io_attach_stdin	-2.388e+02	5.422e-08	-3.360e+03	~ 0
io_attach_stdout	6.461e-01	9.879e-01	6.049e+02	4.399e-05
io_tty	8.930	8.321e-01	2.250e+03	1.038e-39
cmd_envvars	2.471e+01	1.561e-84	7.063	3.284e-07
cmd_args	-7.068e+01	4.092e-23	-7.686e+01	4.904e-28
fs_volumes	8.365e+01	6.614e-17	7.892e+01	1.886e-15
fs_size	7.012e-08	~ 0	3.571e-08	2.261e-27
fs_layers	3.109e+01	~ 0	2.576e+01	~ 0
net_ports	1.915e+01	2.027e-05	-1.843e+01	4.323e-05

We start our analysis with linear regression techniques, as these give us explainable results about which image factors influence the start times [23]. Note that we do not aim to find the best start time prediction model in this paper; we instead investigate which parameters impact the start time. We first consider the influence of our image configuration parameters on the start time individually; that is, we consider a univariate regression problem. Therefore, we train a linear regression model in the form

$$y = \beta_0 + \beta_1 x. \tag{2}$$

Here, y denotes the start time, our dependent variable, and x the considered feature, as the independent variable, while β_0 and β_1 are the linear regression coefficients. In addition, we consider the p-value, which indicates the probability of observing similar results, under the assumption that the null hypothesis is correct. Table IV shows the β_1 and p-values of different features from our dataset. In the table, p-values smaller than 1e-100 were rounded to zero.

The values in Table IV for the univariate regression show that all factors considered, except *io_attach_stdout* and *io_tty*, yield results with a low p-value in the linear regression. The values' units play a role in the interpretation of β_1. The start time in our dataset is given in milliseconds, and the size is in bytes. Consequently, the small absolute value of β_1 for the image size is explained. However, a deeper interpretation of the coefficients is unnecessary because the error for the univariate regression is very high for all features. The lowest mean absolute error (MAE) is 777 ms for the feature *fs_layers*. To calculate the MAE, we used a five-fold cross-validation. For comparison, the MAE of a baseline model that predicts the mean of all measurements is 806 ms. This shows that the data do not have a simple linear dependence. This is also confirmed by Figure 5, which shows the scatter plots for the two features with the lowest p-values, *fs_size* and *fs_layers*. Intuitively, one could assume a clear relationship between those features and the start time. However, the figure clearly shows that none of the two factors can explain the variance in the start time individually.

We deduce that no single dominant feature determines the start time; it is instead a multivariate problem. For this rea-

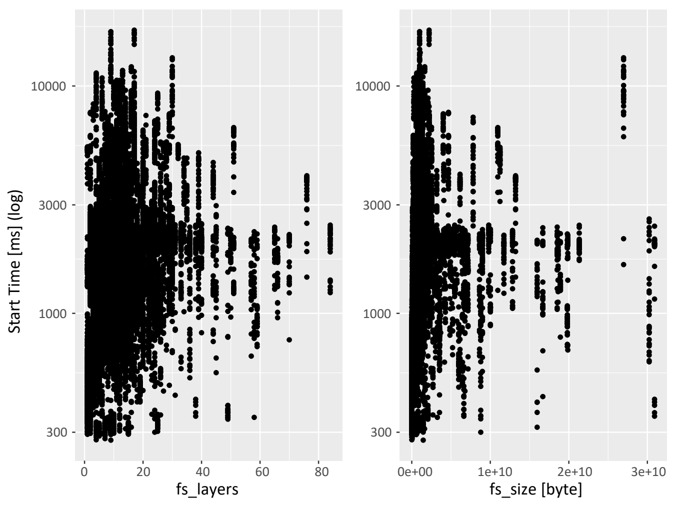

Fig. 5. Dependence between two selected features and the start time.

son, we extend our analysis to multivariate linear regression. Table IV also shows the β_1 coefficients and the p-values of multivariate linear regression. The p-values show that all features individually contribute to the start time prediction in the multivariate regression. Further, we performed an F-test to evaluate whether the features taken together contribute to the start time prediction [23]. The result is a p-value of 2.2e-16, indicating that feature interactions also contribute to the start time prediction and that the model cannot be reduced to a naive intercept-only model. However, the MAE of 772 ms also shows that the multivariate linear model cannot describe the data well, performing just slightly better than the baseline. We conclude that there is no linear relationship between our features and start time.

Due to the complex relationships in the data, we next apply a random forest model as an example of a non-linear explainable regression approach. We build the random forest with 500 trees, having each three randomly sampled features and allowing unlimited depth. Table V shows the feature importance values for the Google Cloud environment retrieved from the random forest in the second column. We see that size is the most important feature, followed by the number of root file system layers. The MAE of the model based on a five-fold cross-validation is 329 ms, which is less than half the error of the linear models and the baseline. This shows that non-linear models can better describe the data. However, searching for the best prediction model is out of the scope of this paper and left as future work as it requires a broader analysis.

C. Can the results be confirmed in a second test environment?

The results from the previous section captured our measurement results in the Google Cloud environment. In the following, we analyze to what extent our findings apply in the self-hosted environment. As stated in Section V-A, the start times measured in our self-hosted environment have a higher variance than those measured on Google Cloud VMs. Furthermore, we note that of the 1008 images tested, 1007

TABLE V
RANDOM FOREST FEATURE IMPORTANCE VALUES FOR BOTH TEST ENVIRONMENTS.

Feature	Feature Importance	
	Google Cloud	Self-hosted
io_attach_stdin	0.007	0.008
io_attach_stdout	0.009	0.011
io_tty	0.008	0.010
cmd_envvars	0.123	0.155
cmd_args	0.048	0.032
fs_volumes	0.040	0.032
fs_size	0.527	0.554
fs_layers	0.176	0.165
net_ports	0.062	0.033

start faster in the Google Cloud environment, as measured by median start times, than in the self-hosted environment. On average, an image starts 6.1 seconds faster in the Google Cloud than in the self-hosted environment. From this, we conclude that the environment significantly influences the absolute start time values.

In the following, we investigate whether our statements about the impacts of different features on start time are confirmed in our second environment. Figure 6 shows, for the same nine features, the median start time for all sample images in both environments as a function of feature value. We see that the absolute numbers of start times differ. However, the distribution of data points is clearly similar in both test environments, indicating that each of the nine features has a similar impact in both environments. Apart from a few outliers, similar trends are visible for nearly all of the selected features.

To compare the results quantitatively, Table VI shows different error measures for all evaluated models in our two test environments. All error measures have been derived using five-fold cross-validation. The baseline model is a model that always predicts the mean of the training data. For the univariate linear models, the error depends on the feature selected for the regression; we report the best score achieved. As the MAE is an absolute measure and differs significantly for the two environments, we also present the mean absolute percentage error (MAPE) as a relative measure to eliminate the scale of the start time. As the MAPE shows, the prediction quality is roughly the same considering the pairwise comparison in both environments. We conclude that the environment determines the absolute values of the start time. However, from Figure 6 and the comparable prediction quality of the models, we conclude that the influence of the image configuration parameters is comparable in both environments. This is also confirmed by the feature importance of the random forest model shown in the third column of Table V. We see that size, layers, and the number of environment variables were the most relevant features in both environments, while all other features were significantly less important for the prediction.

D. Summary

In the following, we summarize our main findings. First, we found that container start times vary significantly in the same

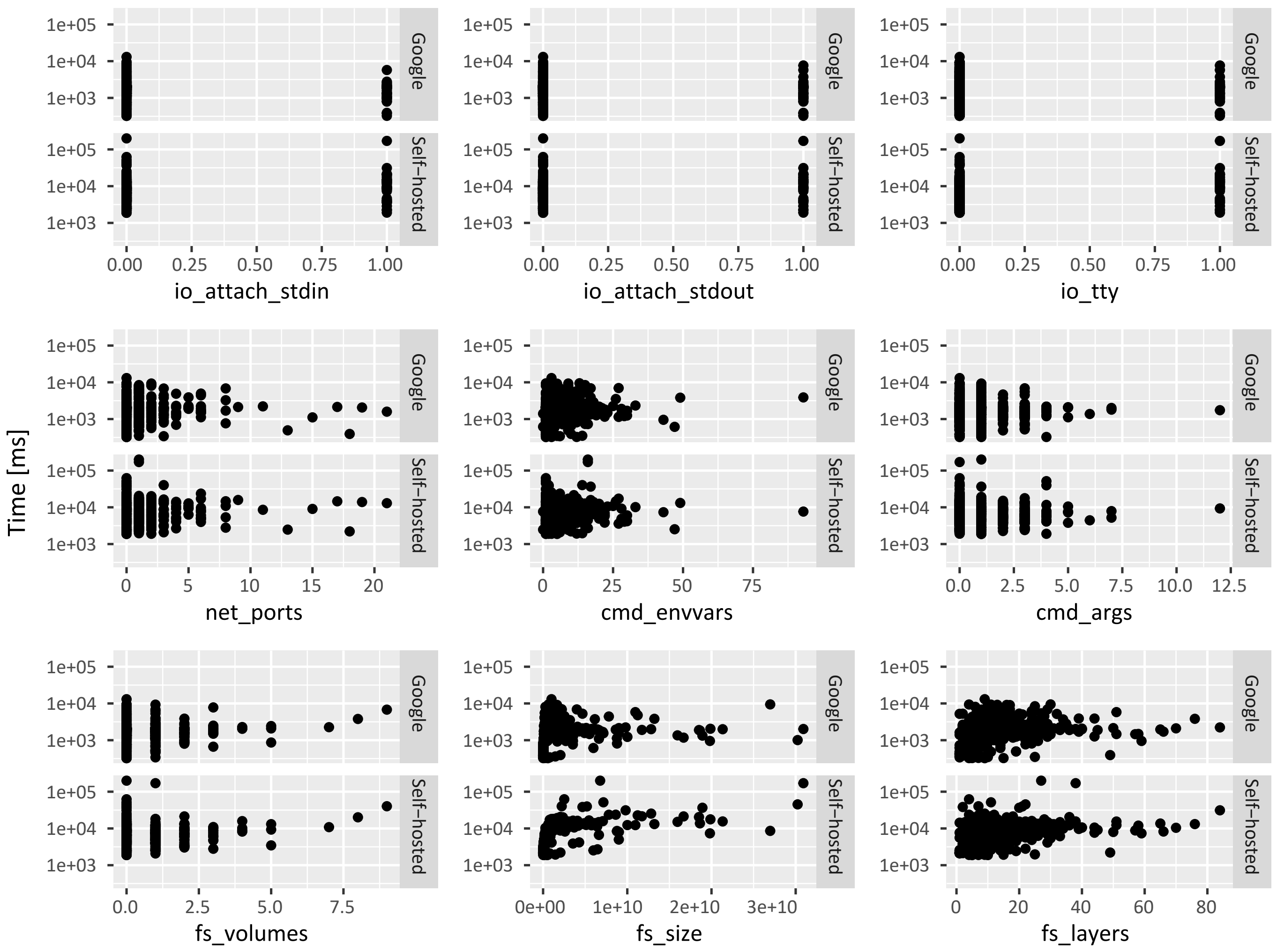

Fig. 6. Measured start times, in ms, as a function of nine different features in each of the Google and self-hosted test environments.

TABLE VI
ERROR MEASURES FOR ALL MODELS AND TEST ENVIRONMENTS.

Model	**Google Cloud**		**Self-hosted**	
	MAE [ms]	MAPE	MAE [ms]	MAPE
Baseline	806	0.607	4513	0.748
Univariate LinReg	$\geq$ 777	$\geq$ 0.565	$\geq$ 3751	$\geq$ 0.569
Multivariate LinReg	772	0.559	3661	0.541
Random Forest	327	0.215	1816	0.205

environment. For example, in Google Cloud, the start time between different image configurations varies between 277 ms and 17.6 s. Furthermore, we could quantify the variance when starting a container image. The median coefficient of variation was 15.3% in the Google Cloud and 17.7% in the self-hosted environment. Only a few outliers had significantly larger CoVs. We showed that modeling of start times cannot be reduced to the examination of one single feature and that a multivariate approach is necessary. We also showed that the relationship between image configuration parameters and the start time is not linear. Using the random forest as a non-linear regressor, we achieved significantly better prediction results than with linear models.

We confirmed all of these statements in both test environments. From the similar feature importance of the random forest models and the comparison of the start times mapped to the features, we infer that image configuration parameters have a comparable influence on start times in each environment. However, absolute start time values are strongly influenced by host hardware. Thus hardware and software stacks must be modeled to build an accurate start time prediction model for different environments. Moreover, our results indicate that, for example, the storage technology influences the start times' variability.

VI. Discussion

We next discuss the limitations and open challenges of this work. Our dataset reflects only a subset of all container images and has been collected from one registry. Furthermore, we limit our study to Linux images with amd64 target architecture and start them using one specific version of Docker; we did not investigate whether these results generalize to other operating systems, architectures, and container engines. Nevertheless, we argue that our results remain valuable, as the selected technologies are common in modern cloud environments. For technical, time, and cost reasons, we tested only a subset of the more than 200,000 images in our dataset and used their default configuration only. Through our stratified sampling strategy, we aim to reflect the diversity of the dataset in the evaluated sample, but further measurements of other image configurations may provide additional insights.

Previous work has shown that measurements in cloud environments can exhibit considerable variability [24], [25]. To reduce the impact of such variability on our results, we repeated each measurement 30 times for each image, using the randomized multiple interleaved trials methodology, as suggested by state-of-the-art benchmarking guidelines [21], [26]. However, further measurements in the Google Cloud (e.g., in other compute regions) may lead to different results. Furthermore, our measurements were designed in a way that only one container was active on one machine at a time. In practice, several containers usually run in parallel on a VM or server. This and other factors are out of the scope of this work.

As introduced in Section II, the start time, as considered in this paper, cannot be equated with the readiness time. A generalization is impossible due to the application-specific setup and network-specific pull times. In particular, analyzing setup time requires knowledge about the content and purpose of the container image. In this work, we maintained a black-box view on the container's file system. Nonetheless, we showed that the start time has to be considered, as it varies significantly for different container images. Moreover, one has to regard that some image features, especially size, might also influence the pull and setup times. In summary, this paper is a starting point for further measurement-driven investigations of container start times. Some of the limitations in the measurement setup can be eliminated in future work if more resources are available.

VII. Related Work

Faster start times have always been a goal for developers of container technologies. Several studies compare container start times and further performance metrics with other virtualization technologies. Tesfatsion et al. [10] examine VM startup times compared to Docker and Linux containers. Another virtualization technique often used for comparison is unikernels [9], [11], [27], [28]. In summary, all studies conclude that containers have faster start times than VMs. Regarding unikernels, the results are less clear, but the use cases of unikernels are usually limited to single-process applications.

Another group of papers deals with certain characteristics of the nodes used for container deployment and their impact on start times. In general, container start times in different technology stacks can differ significantly. De Velp et al. [29] find that the used storage driver, in particular, significantly impacts the start times. This result is also confirmed by Harter et al. [30], who investigate I/O patterns during startups and propose a specialized storage driver. Lingayat et al. [31] and Mavridis and Karatza [32] quantify overheads introduced by starting containers on virtual machines instead of bare-metal servers. Wei et al. [33] investigate the start times of Kubernetes pods and show the dependency on the overall system load.

Another research area looks at start times concerning different container runtimes, often in combination with security and isolation properties. Kumar et al. [34] compare *runc*, the default Docker runtime, with *Kata*, a runtime with increased isolation and security mechanisms. Others compare different combinations of container managers and runtimes [35], [36]. All works find a clear trade-off between performance and increased security.

Due to the observed start time variabilities, researchers found early optimization potential for start times. Thereby, the optimization approaches focus on specific domains and workloads. One major use case is serverless computing, where optimization approaches include caching techniques, faster installation of dependencies, and container reuse [37], [38]. Ahmed and Pierre [7] find that pull times have an essential impact in edge computing use cases and propose optimization through better download schemes and layer decompressing. Littley et al. [39] find optimization potential in container registries intending to reduce pull times.

There are few prior empirical studies of large sets of container images. Zhao et al. [40] use a large Docker Hub dataset to explore the properties of layers, image popularity, compressed and uncompressed image sizes, and what type of files the analyzed containers include. In contrast to our work, start times are not investigated. Cito et al. [41] analyze over 70,000 Dockerfiles from GitHub and identify common base images, primary programming languages, and more. The ImageJockey framework of Yoshimura et al. [42] enables performance testing of multiple container images. The uniqueness and novelty of our study lie in the interconnection of a large container dataset with start times. Furthermore, in contrast to related work, we maintain a black-box view of container execution and content, relying only on data available before starting the container.

VIII. Conclusion

Modern containerized cloud, serverless, and edge applications require high performance, scalability, and resilience. All these characteristics are inevitably tied to low container start times. We have reported the most extensive empirical study to date on how container image configurations impact start times. We analyzed the configuration parameters of over 200,000 open-source Docker Hub images and showed the high diversity of the dataset. On a sample of this dataset, we ran a total of 60,480 container starts in two test environments and analyzed the impact of image configuration parameters on the start time.

The main finding of our study is that no single configuration parameter determines the start time of a container; the start time rather depends on many variables. Our results show that different image configurations have significantly different start times in one environment. We could confirm the findings on a second test environment to a certain extent. However, we found that the environment strongly influences the absolute values of the start times.

This work offers numerous starting points for future work. One important question is to what extent we can predict the start time of a container for a specific environment without having started it before. This question builds directly on this work since the image configuration parameters that we discussed here and hope to use in making such predictions can be obtained before a container is launched. We showed that hardware parameters have to be modeled for an accurate prediction; however, as these parameters are known and relatively stable for one environment, creating a suitable model is feasible. Another promising research topic is the relationship between start and readiness times. Both research directions promise to benefit real cloud, serverless, and edge environments and to provide valuable inputs for simulations and models.

Acknowledgments

This paper is based upon work supported by the Google Cloud Research Credits program with the award GCP213506809. The authors thank Oliver Wizemann and Nicholas Erhard, who assisted in this project as student researchers.

References

[1] T4 Labs Inc. (2020) Container platform market share, market size and industry growth drivers, 2018–2023. https://www.t4.ai/industries/container-platform-market-share.

[2] Cloud Native Computing Foundation (CNCF). (2022) CNCF annual survey 2021. https://www.cncf.io/reports/cncf-annual-survey-2021/.

[3] M. Carter. (2021) Docker index shows momentum in developer community activity. https://www.docker.com/blog/.

[4] X. Liu, J. Wen, Z. Chen, D. Li, J. Chen, Y. Liu, H. Wang, and X. Jin, "Faaslight: General application-level cold-start latency optimization for function-as-a-service in serverless computing," *ACM Trans. Softw. Eng. Methodol.*, 2023.

[5] Z. Li, L. Guo, Q. Chen, J. Cheng, C. Xu, D. Zeng, Z. Song, T. Ma, Y. Yang, C. Li, and M. Guo, "Help rather than recycle: Alleviating cold startup in serverless computing through inter-function container sharing," in *USENIX Annual Technical Conference*, 2022, pp. 69–84.

[6] S. Hu, W. Shi, and G. Li, "CEC: A containerized edge computing framework for dynamic resource provisioning," *IEEE Transactions on Mobile Computing*, 2022.

[7] A. Ahmed and G. Pierre, "Docker container deployment in fog computing infrastructures," in *IEEE International Conference on Edge Computing*, 2018, pp. 1–8.

[8] M. Straesser, J. Grohmann, J. von Kistowski, S. Eismann, A. Bauer, and S. Kounev, "Why is it not solved yet? challenges for production-ready autoscaling," in *ACM/SPEC International Conference on Performance Engineering*, 2022, p. 105–115.

[9] B. Xavier, T. Ferreto, and L. Jersak, "Time provisioning evaluation of KVM, Docker and unikernels in a cloud platform," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2016, pp. 277–280.

[10] S. K. Tesfatsion, C. Klein, and J. Tordsson, "Virtualization techniques compared: Performance, resource, and power usage overheads in clouds," in *ACM/SPEC International Conference on Performance Engineering*, 2018, p. 145–156.

[11] R. Schmoll, T. Fischer, H. Salah, and F. H. P. Fitzek, "Comparing and evaluating application-specific boot times of virtualized instances," in *2nd IEEE 5G World Forum*, 2019, pp. 602–606.

[12] Open Container Initiative. (2022) Image format specification. https://github.com/opencontainers/image-spec/.

[13] Sysdig Inc. (2021) 2021 container security and usage report. https://sysdig.com/blog/sysdig-2021-container-security-usage-report/.

[14] Open Container Initiative. (2022) Runtime specification. https://github.com/opencontainers/runtime-spec/.

[15] Docker Docs. (2022) Docker run reference. https://docs.docker.com/engine/reference/run/.

[16] The Moby Project. (2022) Layer store (in-code documentation). https://github.com/moby/moby/blob/master/layer/layer_store.go.

[17] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[18] C.-E. Särndal, B. Swensson, and J. Wretman, *Model Assisted Survey Sampling*. Springer Science & Business Media, 2003.

[19] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[20] A. C. Rencher and M. Schimek, "Methods of multivariate analysis," *Computational Statistics*, vol. 12, no. 4, pp. 422–422, 1997.

[21] A. Abedi and T. Brecht, "Conducting repeatable experiments in highly variable cloud computing environments," in *8th ACM/SPEC International Conference on Performance Engineering*, 2017, p. 287–292.

[22] Google Cloud Compute Docs. (2023) Benchmarking persistent disk performance. https://cloud.google.com/compute/docs/disks/benchmarking-pd-performance.

[23] S. A. Glantz and B. K. Slinker, *Primer of Applied Regression & Analysis of Variance*. McGraw-Hill, Inc., New York, 2001, vol. 654.

[24] P. Leitner and J. Cito, "Patterns in the chaos—A study of performance variation and predictability in public IaaS clouds," *ACM Transactions on Internet Technology*, vol. 16, no. 3, 2016.

[25] C. Laaber, J. Scheuner, and P. Leitner, "Software microbenchmarking in the cloud. How bad is it really?" *Empirical Software Engineering*, vol. 24, no. 4, p. 2469–2508, 2019.

[26] S. Kounev, K.-D. Lange, and J. von Kistowski, *Systems Benchmarking*, 1st ed. Springer International Publishing, 2020.

[27] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, and F. De Turck, "Unikernels vs containers: An in-depth benchmarking study in the context of microservice applications," in *IEEE 8th International Symposium on Cloud and Service Computing*, 2018, pp. 1–8.

[28] T. Goethals, M. Sebrechts, M. Al-Naday, B. Volckaert, and F. De Turck, "A functional and performance benchmark of lightweight virtualization platforms for edge computing," in *IEEE International Conference on Edge Computing and Communications*, 2022, pp. 60–68.

[29] G. E. de Velp, E. Rivière, and R. Sadre, "Understanding the performance of container execution environments," in *6th International Workshop on Container Technologies and Container Clouds*, 2021, p. 37–42.

[30] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy Docker containers," in *14th USENIX Conference on File and Storage Technologies*, 2016, pp. 181–195.

[31] A. Lingayat, R. R. Badre, and A. Kumar Gupta, "Performance evaluation for deploying Docker containers on baremetal and virtual machine," in *3rd International Conference on Communication and Electronics Systems*, 2018, pp. 1019–1023.

[32] I. Mavridis and H. Karatza, "Performance and overhead study of containers running on top of virtual machines," in *IEEE 19th Conference on Business Informatics*, vol. 02, 2017, pp. 32–38.

[33] T. Wei, M. Malhotra, B. Gao, T. Bednar, D. Jacoby, and Y. Coady, "No such thing as a "free launch"? Systematic benchmarking of containers," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2017, pp. 1–6.

[34] R. Kumar and B. Thangaraju, "Performance analysis between RunC and Kata container runtime," in *IEEE International Conference on Electronics, Computing and Communication Technologies*, 2020, pp. 1–4.

[35] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, "Performance evaluation of container runtimes," in *10th International Conference on Cloud Computing and Services Science*, 2020, pp. 273–281.

[36] W. Viktorsson, C. Klein, and J. Tordsson, "Security-performance trade-offs of Kubernetes container runtimes," in *28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2020, pp. 1–4.

[37] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "SOCK: Rapid task provisioning with serverless-optimized containers," in *USENIX Annual Technical Conference*, 2018, pp. 57–70.

[38] S. Qin, H. Wu, Y. Wu, B. Yan, Y. Xu, and W. Zhang, "Nuka: A generic engine with millisecond initialization for serverless computing," in *IEEE International Conference on Joint Cloud Computing*, 2020, pp. 78–85.

[39] M. Littley, A. Anwar, H. Fayyaz, Z. Fayyaz, V. Tarasov, L. Rupprecht, D. Skourtis, M. Mohamed, H. Ludwig, Y. Cheng, and A. R. Butt, "Bolt: Towards a scalable Docker registry via hyperconvergence," in *IEEE 12th International Conference on Cloud Computing*, 2019, pp. 358–366.

[40] N. Zhao, V. Tarasov, H. Albahar, A. Anwar, L. Rupprecht, D. Skourtis, A. S. Warke, M. Mohamed, and A. R. Butt, "Large-scale analysis of the Docker Hub dataset," in *IEEE International Conference on Cluster Computing*, 2019, pp. 1–10.

[41] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An empirical analysis of the Docker container ecosystem on GitHub," in *IEEE/ACM 14th International Conference on Mining Software Repositories*, 2017, pp. 323–333.

[42] T. Yoshimura, R. Nakazawa, and T. Chiba, "ImageJockey: A framework for container performance engineering," in *IEEE 13th International Conference on Cloud Computing*, 2020, pp. 238–247.

Supplementary Artifacts

We provide two peer-reviewed artifacts complementary to this paper. The first artifact is the dataset containing 200,986 images with their configuration features. The artifact has been uploaded as a standalone artifact on Zenodo to facilitate further studies in the area of application containers. The second artifact reproduces the results, tables, and figures from this paper. For this purpose, we provide a CodeOcean capsule that processes our measurement results and reproduces all quantitative results in this paper. Both artifacts contain detailed descriptions and documentation on their respective platforms.

A. Image Dataset

The image dataset described in Section III of this paper is available on Zenodo as a standalone artifact:

https://doi.org/10.5281/zenodo.7602500

Similar to Table I of the paper, we explain the features of the 200,986 images in the dataset and provide information about the acquisition of the dataset on the website. The dataset is compressed and in CSV format and can therefore be easily analyzed in any programming language. In the future, the dataset could be used for further studies on open-source container images.

B. Processing Scripts

The processing scripts and measurement results from all test environments are available in a CodeOcean capsule with the following link:

https://doi.org/10.24433/CO.4595026.v2

The scripts are a mix of Python and R scripts. The CodeOcean capsule offers a pre-configured execution environment with all required libraries installed. The scripts use the image dataset (same as uploaded on Zenodo) and the start time measurement results from the self-hosted and Google Cloud environments as inputs. Documentation on the processing steps can be found in the script files. A complete run takes about 13 minutes, where most of the time is spent creating the Random Forest regression models. If the reproducible run feature of CodeOcean is used, Tables III, IV, V, and VI are obtained in text form. Moreover, Figures 2, 3, 5, and 6 from the paper are created.

As referenced in Section IV-B, we provide additional information on the sampling process. We list the algorithms and hyperparameters used for the dataset clustering as part of the stratified sampling process. We report a quantitative breakdown of the feature importance for the final clustering. Last, we provide a visualization of the feature set for the tested sample.