

# Not all Dockerfile Smells are the Same

An Empirical Evaluation of Hadolint Writing Practices by Experts

Giovanni Rosa

Simone Scalabrino

Gregorio Robles

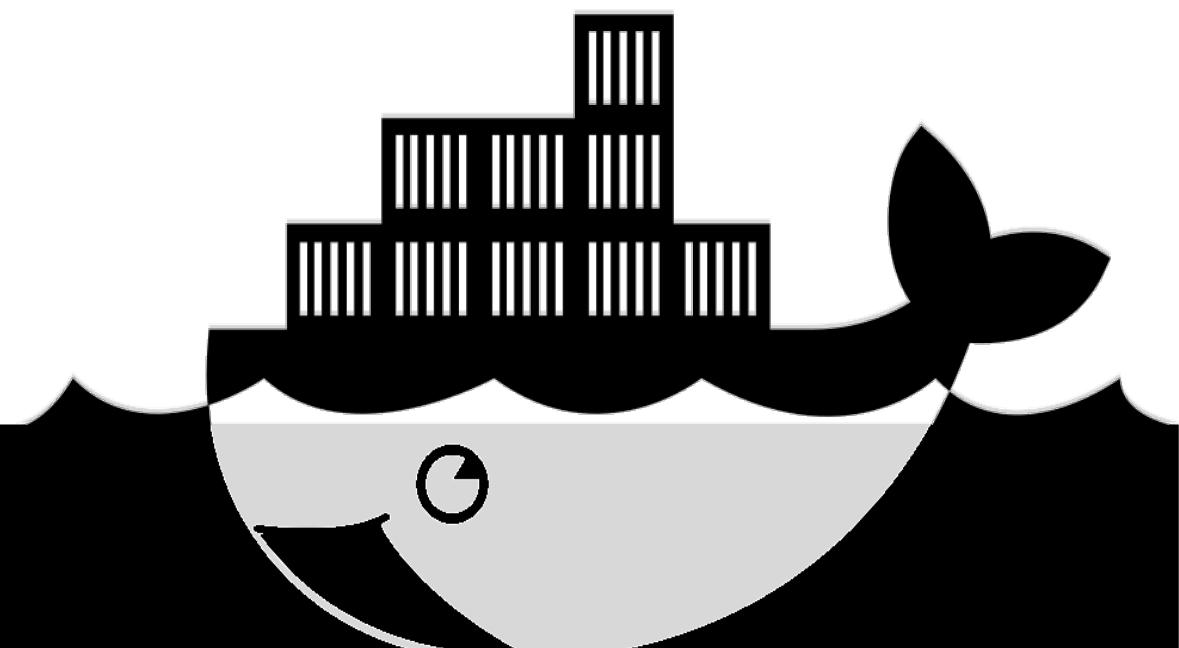
Rocco Oliveto

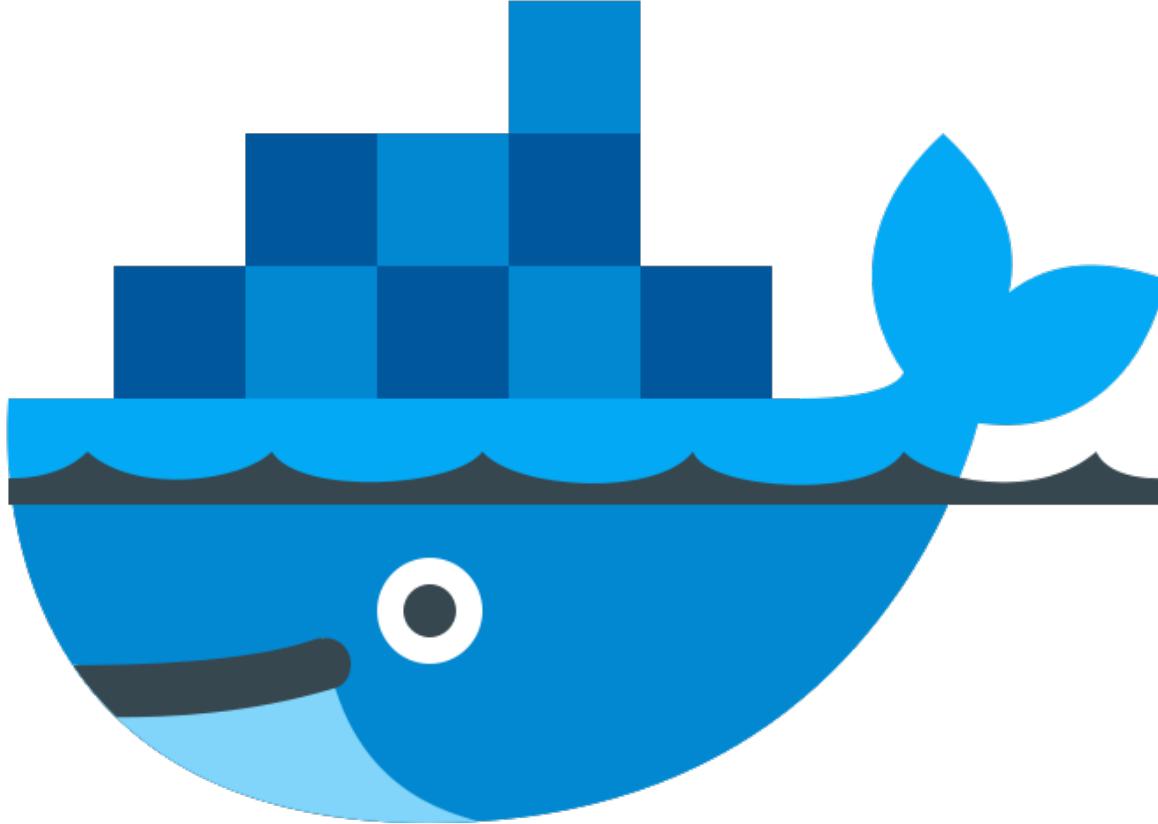


Università degli  
Studi del Molise



Universidad Rey  
Juan Carlos





Docker is the Reference Tool  
for Software Containerization

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY . .
```

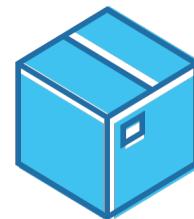
# Dockerfile

Docker in a nutshell

```
1 FROM node:12-alpine  
2  
3 RUN apk add --no-cache python2 g++ make  
4  
5 WORKDIR /app  
6 COPY . .
```

Dockerfile

→  
build



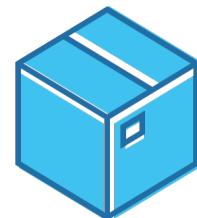
Image

Docker in a nutshell

```
1 FROM node:12-alpine  
2  
3 RUN apk add --no-cache python2 g++ make  
4  
5 WORKDIR /app  
6 COPY . .
```

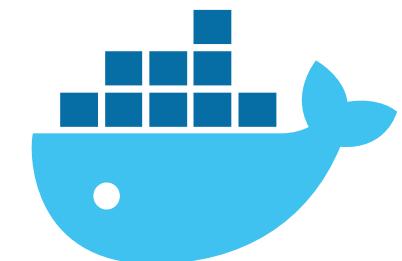
Dockerfile

build



Image

run



Container

Docker in a nutshell



Writing  
Dockerfiles  
is challenging

## Revisiting Dockerfiles in Open Source Software Over Time

Eng et al., 2021

Docker is becoming ubiquitous with its application for developing and deploying applications. Previous studies have analyzed Dockerfiles that are used to create container images in order to better understand how to improve Docker coding. These studies have been conducted over time. In this paper, we aim to fill the gap by revisiting the findings of previous studies using the largest set of Dockerfiles from GitHub with 8.7 million unique Dockerfiles. We also analyze Dockerfiles in the WebAssembly Docker repository spanning from 2013–2020. We conduct a historical view of the Dockerfiles by analyzing the evolution of Dockerfiles. We also use the history of GitHub to analyze the evolution of Dockerfiles. We also re-evaluate previous findings of a dosimeter tool in using its improved version to analyze Dockerfiles. Our results show that Dockerfile smell counts are slightly decreasing, showing that Dockerfile authors are likely getting better at Dockerfile writing. However, we find that Dockerfiles in our previous analysis from prior works have been current in many of their findings and their suggestions to build better code for Dockerfiles have been well-justified.

Index Terms—Docker, Dockerfiles, GitHub, Dockerhub, Docker coding, Docker evolution, Docker smells.

## A Study of Security Vulnerabilities on Docker Hub

Shu et al., 2017

### ABSTRACT

Docker, a tool for creating and running programs in containers consistently across platforms, was initially released to the public on March 30, 2013 [1], [2]. Ever since its initial release, Docker has been rapidly growing, especially following its 3.5 million desktop installations and 7 million Dockerfiles in GitHub. Dockerfiles are widely used to configure Docker images by specifying the steps required to build a Docker image. Dockerfiles are often used to better understand Docker. They encapsulate various images and configurations in Dockerfiles. After 6 years in Docker, there are more than 1.5 million Dockerfiles in GitHub, which can be used to extract Docker images for analysis.

2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR)

## An Empirical Study of Build Failures in the Docker Context

Wu et al., 2021

### ABSTRACT

Docker containers have become the de-facto industry standard. Docker builds often break, and a large amount of efforts are put into troubleshooting broken builds. Thus, studies have evaluated the frequency and causes of Docker build failures. However, it is unclear about the frequency and the effect of failures that occur in Docker builds of open-source projects. This paper presents an empirical study of Docker build failures in the Docker context. We collected Docker build logs from 1,639 open-source projects located on GitHub. Using the Docker build logs, we measure the frequency-of-build-failure and report the result. Furthermore, we compare the evolution of Docker build failures over time. Our results help Docker users to understand Docker build failures and motivate the need for collecting more empirical evidence.

### KEYWORDS

Docker, Build Failure, Open-source

ICSE Refinement: None

This work was partially funded by the National Natural Science Foundation of China (No. 61872401) and the National University Students' Innovation and Entrepreneurship Training Program (No. 201810511014).

✉ wujiayi@ust.hk (Jiayi Wu); jywu@ust.hk (Yi Wu)

✉ jywu@ust.hk

## Revisiting Dockerfiles in Open Source Software Over Time

Eng et al., 2021

**Abstract** Docker is becoming ubiquitous with its application for developing and deploying applications. Previous studies have analyzed Dockerfiles that are used to create container images in order to better understand how to improve Docker coding. These studies have been limited to a single point in time. In this paper, we revisit the findings of previous studies using the largest set of Dockerfiles from GitHub with over 8.7 million unique Dockerfiles. We find that the Dockerfile adoption rate in the WebAssembly ecosystem has increased by 100% over the past decade spanning from 2013–2020. We conduct a historical view of the Dockerfiles in GitHub by examining the evolution of Dockerfiles to see the history of the evolution of Dockerfiles. We also reexamine previous findings of a dosimeter tool to using 105 unique Dockerfiles from GitHub. Our results show that the dosimeter tool identifies Dockerfiles with small errors are slightly decreasing showing that Dockerfile authors are likely getting better at Dockerfile writing. We also find that the dosimeter tool from our previous analysis from prior work has been current in many of their findings and their suggested build better tools for Dockerfile authors.

*Index Terms—Docker, Dockerfiles, GitHub, Docker adoption*

Dockerfiles [6] (2006, 2020) and RUN being the most popular Dockerfile instruction [5].

### II. PREVIOUS WORK

In previous work, researchers have used Dockerfiles to better understand Docker in repositories and in open-source ecosystems.

## A Study of Security Vulnerabilities on Docker Hub

Shu et al., 2017

### ABSTRACT

Docker containers have recently become a popular approach to provision multiple applications over traditional virtual machines. Docker containers are more lightweight than traditional virtual machines and are easier to manage. Dockerfiles are the configuration files used to build Docker containers. They encapsulate various images and configurations and have been used for over 8 years in Docker. Using code analysis, we analyze Dockerfiles in GitHub to better understand them. We also evaluate Dockerfiles on Docker Hub and its relatives, Docker Cloud and Docker Swarm. We find that Dockerfiles on Docker Hub are more complex than Dockerfiles on GitHub. We also find that Dockerfiles on Docker Hub are more likely to contain security issues.

*Index Terms—Docker, Dockerfiles, GitHub, Docker Hub, Docker Cloud, Docker Swarm, security, Dockerfile analysis*

2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR)

## An Empirical Study of Build Failures in the Docker Context

Wu et al., 2021

### ABSTRACT

Docker containers have become the de-facto industry standard. Docker builds often break, and a large amount of efforts are put into troubleshooting broken builds. Thus, studies have evaluated the frequency and impact of build failures. However, there is a lack of research that can provide insights into the causes of build failures. In this paper, we study the causes of build failures in 1,639 open-source projects located on GitHub. Using the Docker build log, we measure the frequency-of-build failures and report the top five reasons. Furthermore, we compare the evolution of Docker build failures. Our results help Dockerfile authors understand Docker build failures and motivate the need for collecting more empirical evidence.

### KEYWORDS

Docker, Build Failure, Open-source

**ACKNOWLEDGMENT**  
This work was supported by the National Natural Science Foundation of China (No. 61872401), the National Key Research and Development Program of China (No. 2018YFB1500300), and the National University Students Innovation Training Program (No. 201910510001).

configuration file (Dockerfile) [2], and host configuration files [4]. These works have emerged a lot of good findings and brought many practical implications to developers, but were not able to fully explain the details of build failures. Therefore, we propose to study the causes of build failures, which motivates the present work. In this paper, we collected Dockerfiles from GitHub and GitHub issues to identify problems early before delivering products to end-users. Specifically, the frequency and impact of build failures have been studied in [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 121

## Revisiting Dockerfiles in Open Source Software Over Time

Eng et al., 2021

**Abstract:** Docker is becoming ubiquitous with containerization for developing and deploying applications. Previous studies have analyzed Dockerfiles to see if certain container images are more popular than others. In this paper, we analyze Dockerfiles used on DockerHub using either Docker Hub or GitHub. In this paper, we revisit the findings of previous studies using DockerHub and GitHub as our source of Dockerfiles. We also compare Dockerfiles found in the World of Code repository spanning from 2013-2018. We reconstruct a historical view of the Dockerfiles found in GitHub and DockerHub. We also review the history to analyze Dockerfiles. We also reconstruct previous findings of a downstream user using 105 Dockerfiles from GitHub and DockerHub. From our analysis, we conclude that Dockerfile sizes seem to slightly decrease over time, while Dockerfile authors are likely getting better at writing Dockerfiles. We also find that Dockerfiles from GitHub are more popular than Dockerfiles from DockerHub. Previous analyses from prior works have been carried in many of their findings and their suggestions to build better Dockerfiles have been partially implemented.

Rui Shu, Xigou Gu and William Enck  
North Carolina State University,  
Raleigh, NC, USA  
**Shu et al., 2017**

**1. INTRODUCTION**

Docker, a tool for containerizing software programs to enable consistency across environments, has quickly become a standard in the Docker on March 26, 2013 [1, 2]. Ever since its release, Docker has accrued a considerable following with over 10 million installations and 7 million Docker Hub users as of May 2014 [3].

The use of container software such as Docker has made application easier to consume, archive, and migrate across platforms. This is achieved by Docker's ability to abstract away the host system by reducing the amount of code needed to configure an appropriate environment by handing the sole configuration instructions to a Dockerfile which can then be used to create images for containers.

2020 IEEE/ACM 11th International Conference on Mining Software Repositories (MSR)  
An Empirical Study of Build Failures in the Docker Container Registry  
Tianyu Wu<sup>1</sup>, Yang Wang<sup>2</sup>, Ming Tang<sup>1</sup>, and Xiang Li<sup>1</sup>  
<sup>1</sup>National University of Defense Technology, China  
<sup>2</sup>University of California, Berkeley, USA  
Wu et al., 2021

**ABSTRACT**  
Rockefeller contractors have become the de-facto industry standard. Rockefeller builds often quickly, and a large amount of efforts are put into building/building better buildings. This research has evaluated the use of what builds in large organizations do. However, the use of time, the frequency and effects of what is built, that occur in the building of buildings. The results of this research will attempt to present a preliminary study on 107,000+ Rockfeller built houses. 1020 open, active projects located on GitHub. Using the 'Rockefeller build' file, we measure the frequency of how often builds and repeat them over time. Furthermore, we explore the evolution of the 'Rockefeller build' file. We also explore the changes that happen to and understand 'Rockefeller build' behaviors and motivate the need for collecting more empirical evidence.

**WORKS**  
Docker Build failure: Open-source

**ACM Reference Format:**  
Wen Wu, Tong Wang, The Wang, and Weizhong Wang. 2000. An Integrated Study of Build Patterns in the Desktop Context. In *11th International Conference on Infocomm Applications (ICIA'2000)*, October 1-3, 2000, Hong Kong, China. ACM, New York, NY, USA, 1 pages. <http://www.cs.toronto.edu/~tongw/>

## 1 INTRODUCTION

blocks to end users from paying compensation fees to access their open platform. It makes the incorporation of software plug-ins into containers and can run on any option [6] since incorporation is [6]. Docker containers have been downloaded 100s times. The

"Annual Consumer Adoption" report<sup>6</sup> found that 79% of companies chose Docker as their primary container technology.

See below how I initially conducted tri-territoriality in my thesis [1].  
[1] https://www.semanticscience.com/paper/sem17-March.pdf  
[2] https://www.semanticscience.com/sem17-conference-submissions/

**Figure 1** The relationship between the number of hours worked per week and the number of hours studied per week.

For responses, comments from [complaints@oig.hrsa.gov](mailto:complaints@oig.hrsa.gov)  
HHS OIG received 3-4-2018. Final Report of Review  
HHS Office of Inspector General (OIG)  
3500 Bryan Avenue, Suite 200, Rockville, MD 20850-3718. (301) 435-0300

<https://doi.org/10.1017/S0008026621000023>

三



# Build Failures

# Security Issues



# ~~Internal Quality Issues~~

## Dockerfile Smells



# **HaDoLint**

**(Haskell Dockerfile Linter)**

A Linter for Dockerfiles

# A Linter for Dockerfiles



## An Empirical Analysis of the Docker Container Ecosystem on GitHub

Jürgen Cito<sup>1</sup>, Philipp Schermann<sup>\*</sup>, John Erik Wittern<sup>1</sup>, Philipp Leitner<sup>1</sup>, Sali Memeti<sup>1</sup>, Michael C. Gall<sup>\*</sup>  
Cito et al., 2017  
<sup>\*</sup> University of Innsbruck and Institute for Software Technology, University of Innsbruck, Austria  
wittern@iis.tugraz.at, gall@iis.tugraz.at

**Abstract**—Docker allows packaging an application with its dependencies into a standardized, self-contained unit (a so-called container), which can be used for software development and to run the application on any system. Dockerfiles are declarative definitions of an environment that aim to enable reproducible builds of the container. They can often be found in source code repositories of open source projects. We conducted an empirical study on the Docker ecosystem, the prevalence of Dockerfiles, quality issues, and the evolution of Dockerfiles. We base our study on a data set of over 70000 Dockerfiles, and contrast this general population with samplings that contain the Top-100 and Top-1000 most popular Docker-using projects. We find that most quality issues (28.6%) arise from missing version pinning (i.e., specifying a concrete version for dependencies). Further, we were able to build 65% of Dockerfiles from scratch from a sample of 560 projects. Integrating quality checks, e.g., to issue version pinning warnings, into the container build process could result into more reproducible builds. The most popular projects change more often than the rest of the Docker population, with 5.81 revisions per year and 5 lines of code changed on average. Most changes deal with dependencies, that are currently stored in a rather unstructured manner. We propose to introduce an abstraction that, for instance, could deal with the intricacies of different package managers and could improve migration to more light-weight images.

**Keywords**—empirical software engineering; GitHub; Docker

### I. INTRODUCTION

Containerization has recently gained interest as a light-weight virtualization technology to define software infrastructure. Containers allow to package an application with its dependencies and execution environment into a standardized, self-contained unit, which can be used for software development and to run the application on any system. Due to their rapid spread in the software development community, Docker containers have become the de-facto standard format [1]. The contents of a Docker container are declaratively defined in a *Dockerfile* that stores instructions to reach a certain infrastructure state [2], following the notion of Infrastructure-as-Code (IaC) [3]. Source code repositories containing Dockerfiles, thus, potentially enable the execution of program code in an isolated and fast environment with one command. Since its inception in 2013, repositories on GitHub have added 70197 Dockerfiles to their projects (until October 2016).

Given the fast rise in popularity, its ubiquitous nature in industry, and its surrounding claim of enabling reproducibil-

ity [4], we study the Docker ecosystem with respect to quality of Dockerfiles and their change and evolution behavior within software repositories. We developed a tool chain that transforms Dockerfiles and their evolution in Git repositories into a relational database model. We mined the entire population of Dockerfiles on GitHub as of October 2016, and summarize our findings on the ecosystem in general, quality aspects, and evolution behavior. The results of our study can inform standard bodies around containers and tool developers to develop better support to improve quality and drive ecosystem change.

We make the following contributions through our exploratory study:

**Ecosystem Overview.** We characterize the ecosystem of Docker containers on GitHub by analyzing the distribution of projects using Docker, broken down by primary programming language, project size, and the base infrastructure (*base image*) they inherit from. We learn, among other things, that most inherited base images are well-established, but heavy-weight operating systems, while light-weight alternatives are in the minority. However, this defeats the purpose of containers to lower the footprint of virtualization. We envision a recommendation system that analyzes Dockerfiles and transforms its dependency sources to work with light-weight base images.

**Quality Assessment.** We assess the quality of Dockerfiles on GitHub by classifying results of a Dockerfile Linter [5]. Most of the issues we encountered considered version pinning (i.e., specifying a concrete version for either base images or dependencies), accounting for 28.6% of quality issues. We also built the Dockerfiles for a representative sample of 560 repositories. 66% of Dockerfiles could be built successfully with an average build time of 145.9 seconds. Integrating quality checks into the “docker build” process to warn developers early about build-breaking issues, such as version pinning, can lead to more reproducible builds.

**Evolution Behavior.** We classify different kinds of changes between consecutive versions of Dockerfiles to characterize their evolution within a repository. On average, Dockerfiles only changed 3.11 times per year, with a mean 3.98 lines of code changed per revision. However, more popular projects revise up to 5.81 per year with 5 lines changed. Dependencies see a high rate of change over time, reinforcing our findings to improve dependency handling from the analysis of the

84%

# Dockerfiles affected by smells

# Detecting Dockerfile Smells

Do developers care?

Do developers care?

Let's ask the experts!



Preliminary  
Study

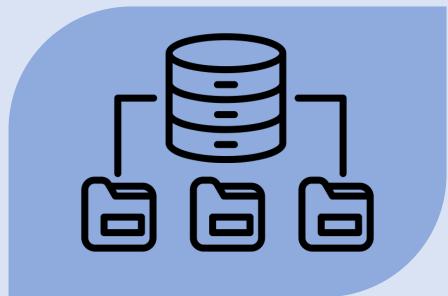
What smells can be  
found in Dockerfiles  
made by experts?



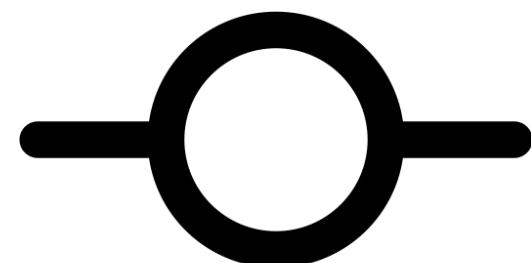
39 Official Docker Repos

Experimental Procedure

# RQ1



Docker Repos



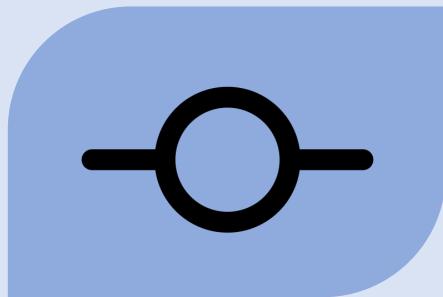
37k Git Commits

Experimental Procedure

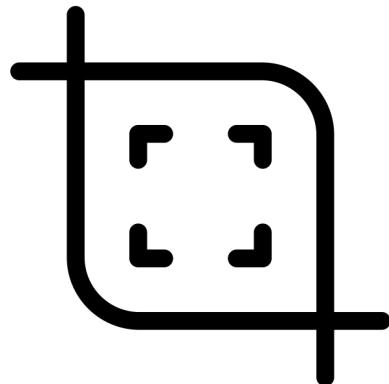
# RQ1



Docker Repos



Repo History



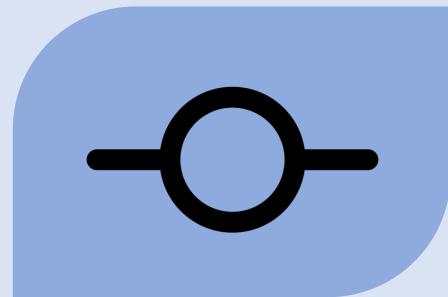
724 Dockerfile Snapshots

Experimental Procedure

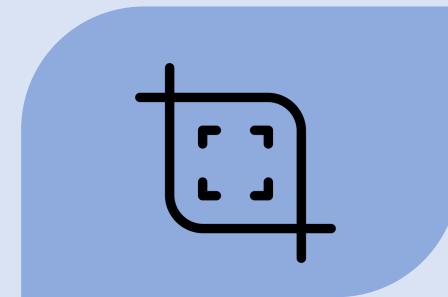
# RQ1



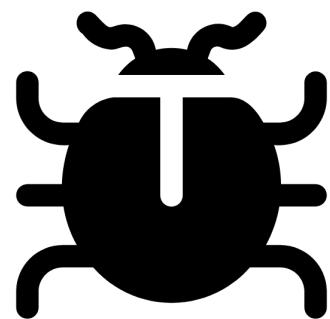
Docker Repos



Repo History



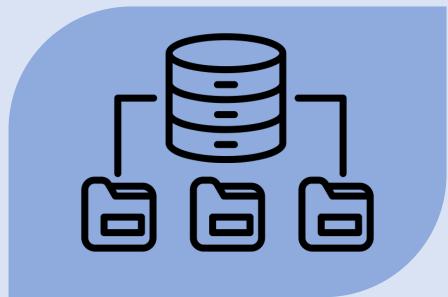
Dockerfiles



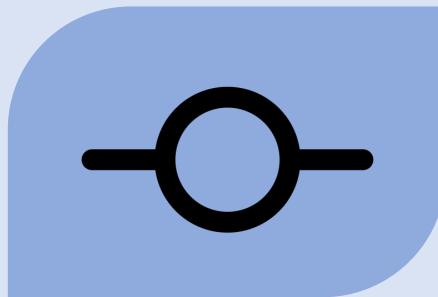
Unique Smell Occurrences

Experimental Procedure

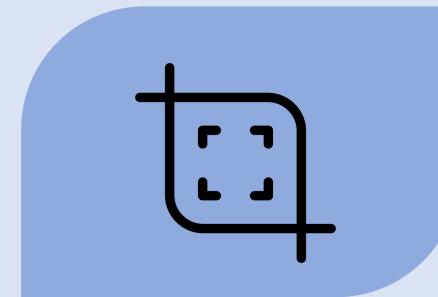
# RQ1



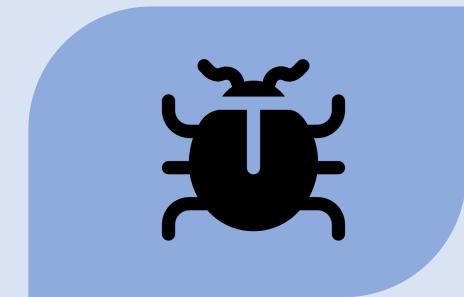
Docker Repos



Repo History



Dockerfiles

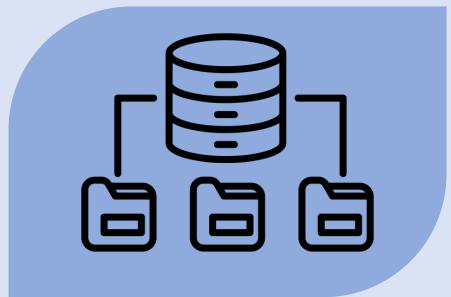


Smells

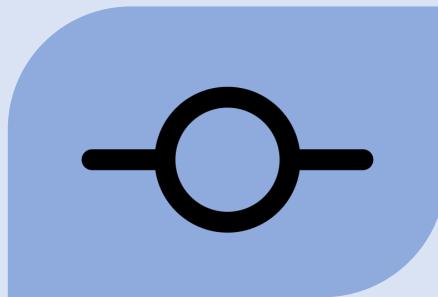
**6k**  
Smells Affecting  
Official Dockerfiles

Results

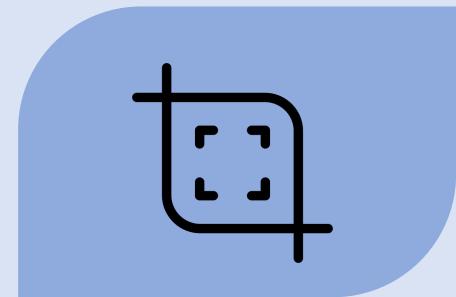
# RQ1



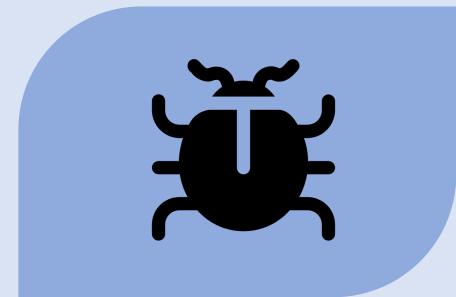
Docker Repos



Repo History



Dockerfiles



Smells

**6k**  
Smells Affecting  
Official Dockerfiles

Use of WORKDIR

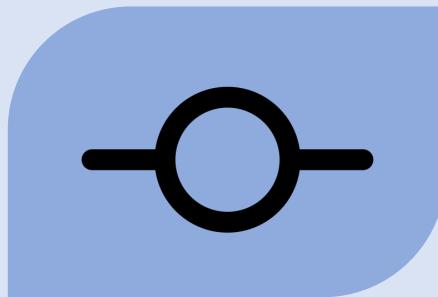
3

Results

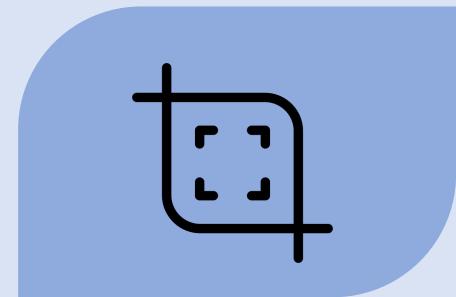
# RQ1



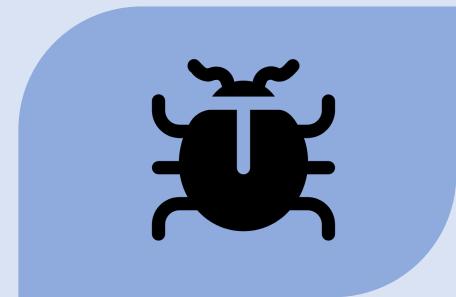
Docker Repos



Repo History



Dockerfiles



Smells

**6K**  
Smells Affecting  
Official Dockerfiles

Use of WORKDIR

3

Version Pinning

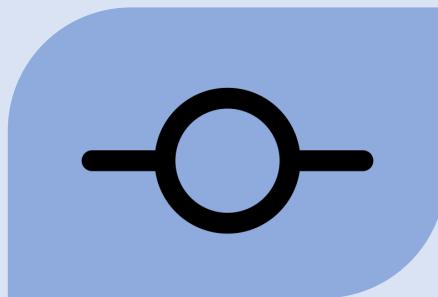
2

Results

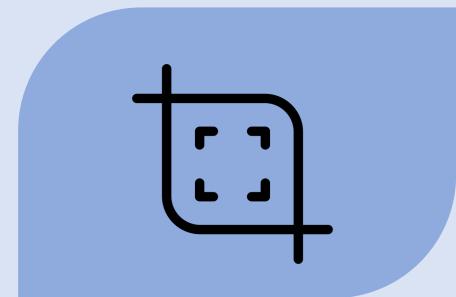
# RQ1



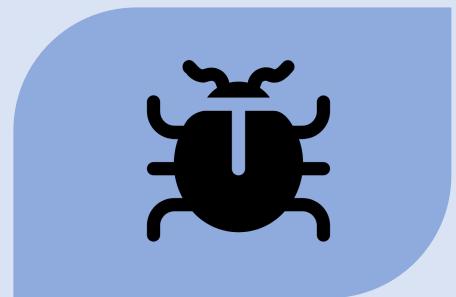
Docker Repos



Repo History



Dockerfiles



Smells

**6K**  
Smells Affecting  
Official Dockerfiles

Use of WORKDIR

3

Version Pinning

2

Shell Pipefail

1

Results



Asking the  
Experts

Are the Dockerfile  
Smells detected by  
Hadolint actual  
Bad Practices?

## RQ2



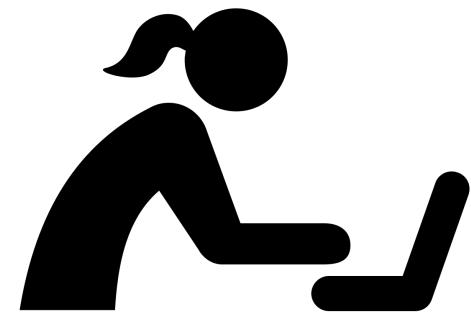
39 Official Docker Repos

Experimental Procedure

## RQ2



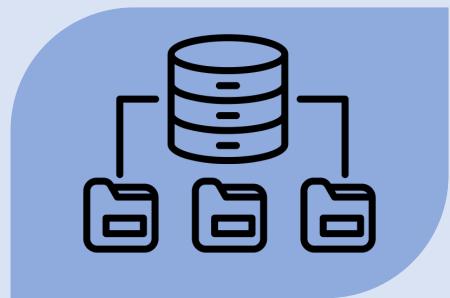
Docker Repos



1k Expert Developers

Experimental Procedure

## RQ2



Docker Repos



Contributors



Demographic Information

Experimental Procedure

## RQ2



Docker Repos



Contributors



Pre-Survey



# Smell Identification Tasks

## Experimental Procedure

# RQ2



Docker Repos



Contributors



Pre-Survey



Identification

1

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY . .
```

Clean Dockerfiles

Experimental Procedure

# RQ2



Docker Repos



Contributors



Pre-Survey



Identification

1

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY ..
```

2

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY ..
```

Clean Dockerfiles

Smell Injection

Experimental Procedure

# RQ2



Docker Repos



Contributors



Pre-Survey



Identification

1

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY ..
```

2

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY ..
```

3

```
1 FROM node:12-alpine
2
3 RUN apk add --no-cache python2 g++ make
4
5 WORKDIR /app
6 COPY ..
```

Clean Dockerfiles

Smell Injection

Identification

Experimental Procedure

## RQ2



Docker Repos



Contributors



Pre-Survey



Identification

**37**  
Participants

Results

## RQ2



Docker Repos



Contributors



Pre-Survey



Identification

**37**  
Participants

**64%**  
Hadolint Smells  
not Identified

Results

## RQ2



Docker Repos



Contributors



Pre-Survey



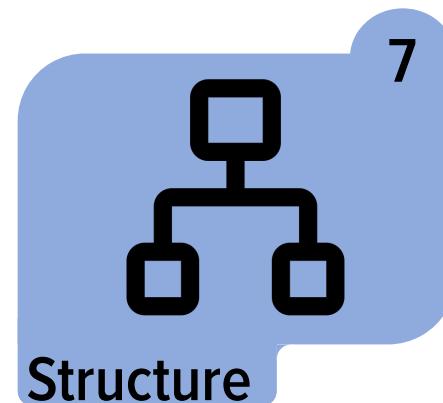
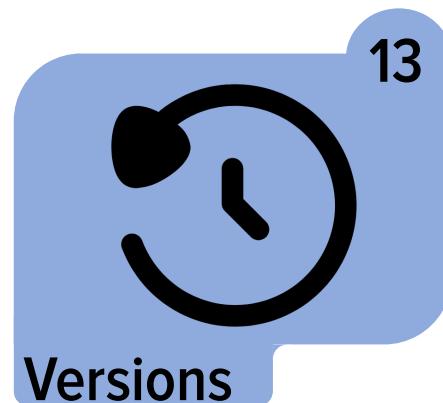
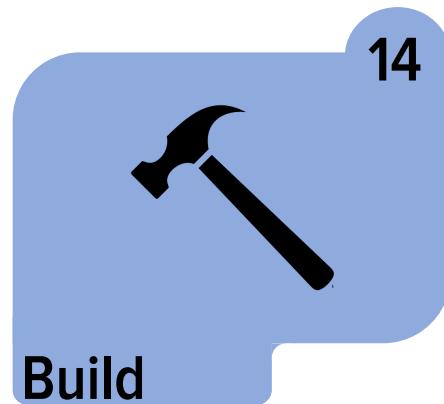
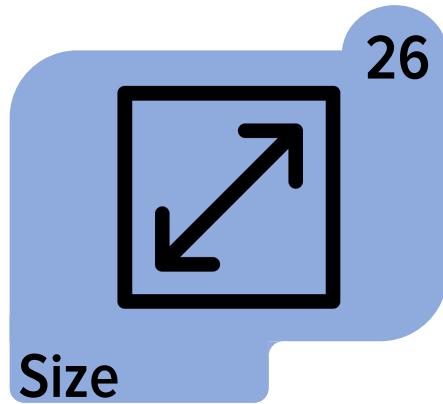
Identification

**37**  
Participants

**64%**  
Hadolint Smells  
not Identified

**26**  
Unexpected  
Smells

Results



# Taxonomy of New Recommendations



Binnacle



DRIVE



Dockercleaner

Mapping with Existing Catalogs



Binnacle



DRIVE



Dockercleaner

20  
Not Mapped

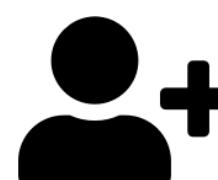
Mapping with Existing Catalogs

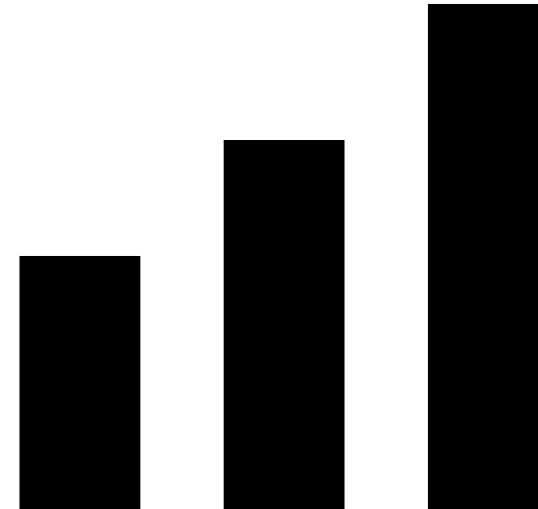
# H Hadolint

 Binnacle

 DRIVE

 Dockercleaner

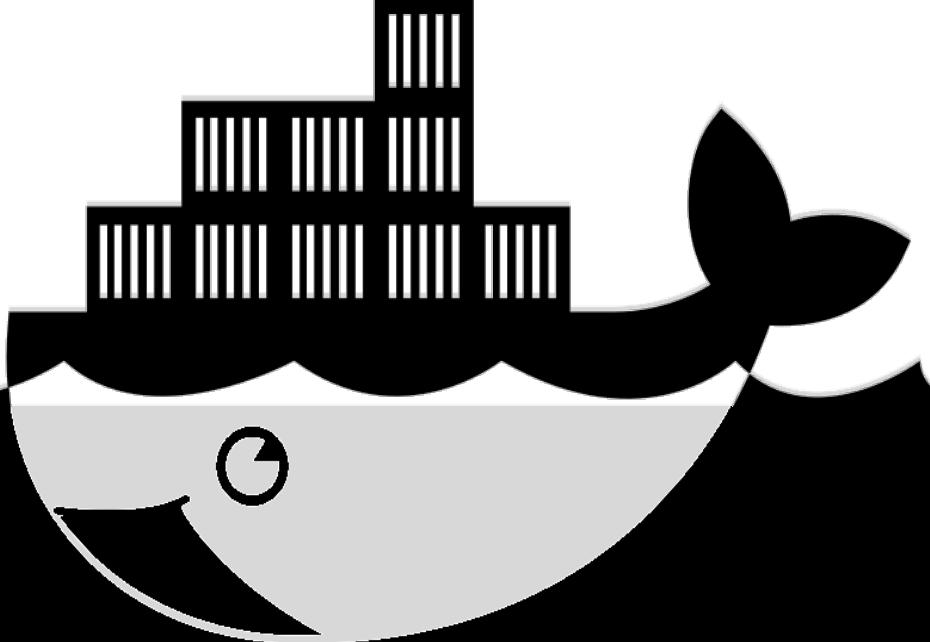
 New smells



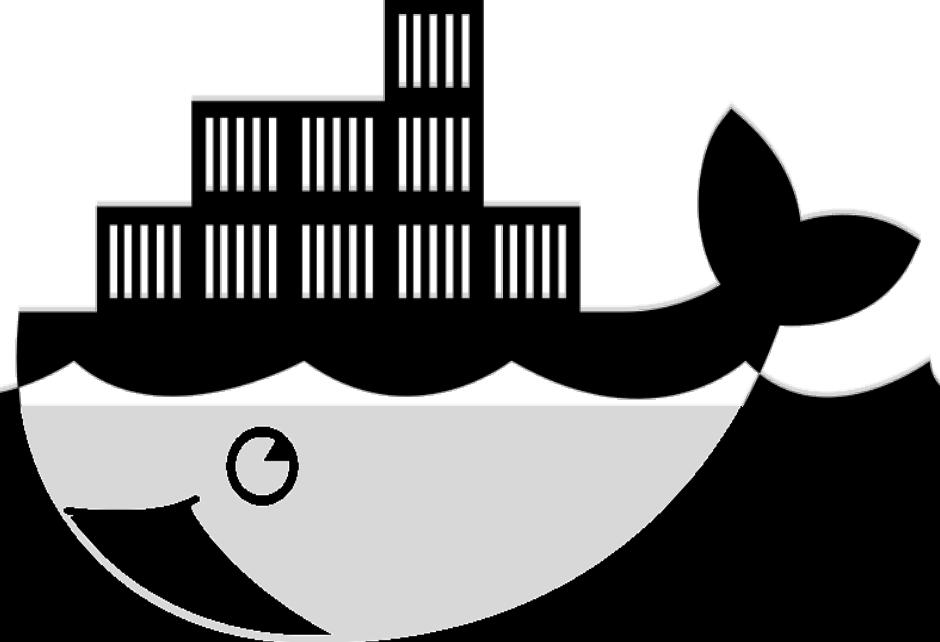
Ranking of  
Dockerfile Smells

Mapping with Existing Catalogs

# Wrapping up

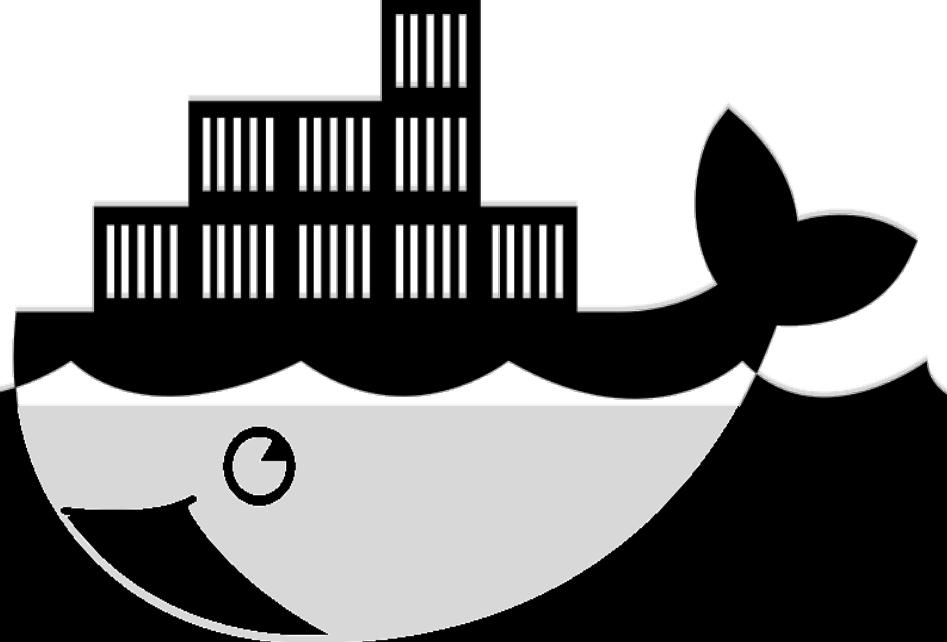


# Wrapping up



- 1 Only a few Hadolint smells are important to experts

# Wrapping up



- 1 Only a few Hadolint smells are important to experts
- 2 Experts focus on performance and security

# Not all Dockerfile Smells are the Same

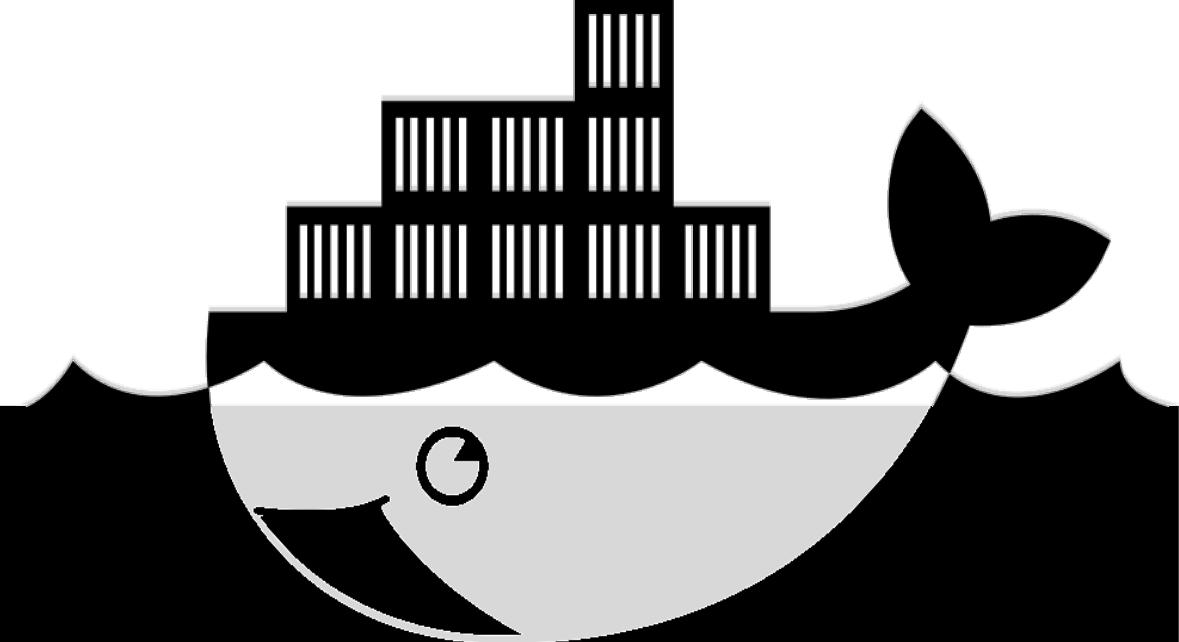
An Empirical Evaluation of Hadolint  
Writing Practices by Experts

Giovanni Rosa

Simone Scalabrino

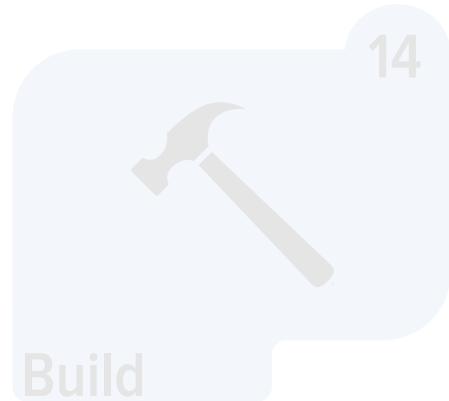
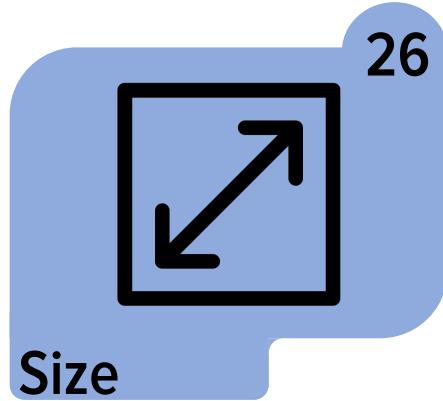
Gregorio Robles

Rocco Oliveto



- 1 Only a few Hadolint smells are important to experts
- 2 Experts focus on performance and security

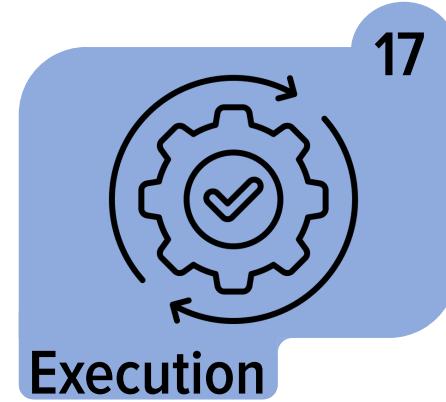




*“Copy only the necessary files from the build context”*



Size



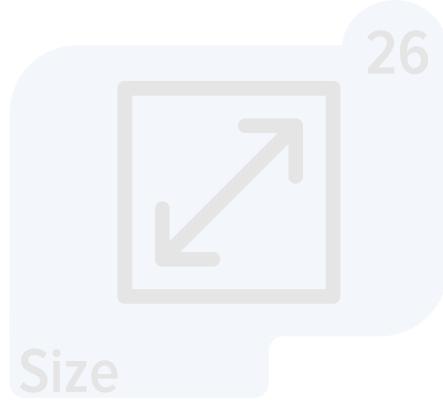
Execution



Build

*“Prefer a binary executable  
for ENTRYPPOINT”*

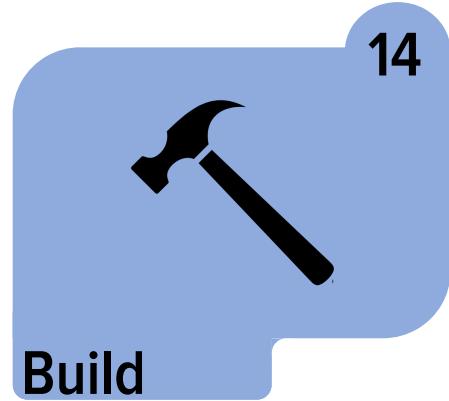
Taxonomy of New Recommendations



Size



Execution



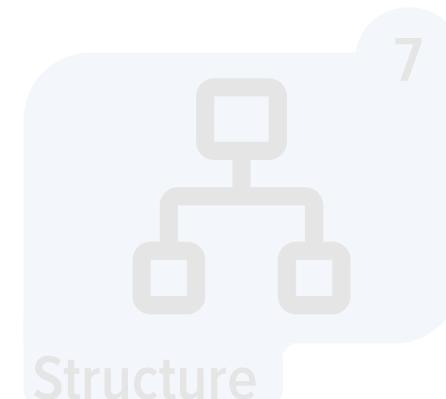
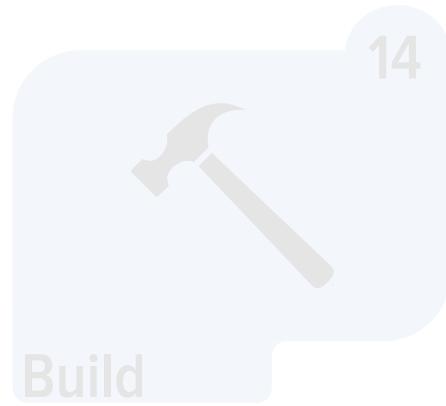
Build

A large, semi-transparent text box containing the quote. Behind it are three smaller, semi-transparent rounded rectangles: one with a camera icon labeled 'Versions' (number 13), one with keyholes labeled 'Structure' (number 7), and one with a chain icon labeled 'Dependencies' (number 1).

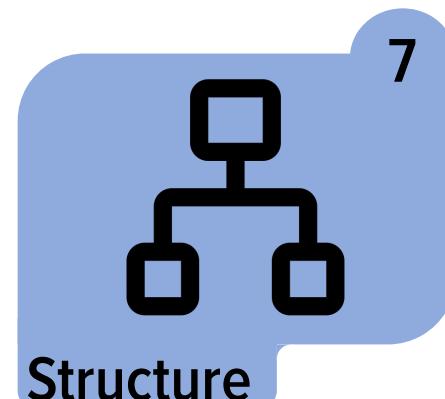
Taxonomy of New Recommendations



“*Prefer popular  
base images*”



*“Avoid hard-coded values”*



Taxonomy of New Recommendations