Technique of reproducing bug and debugging under complex environment

Jun Luo[1, a], Jinhua Li (corresponding author) [2, b]

[1] College of Information Engineer, Qingdao University, Qingdao 266000, China;

[2] College of Information Engineer, Qingdao University, Qingdao 266000, China.

[a]luojun96@live.cn, [b]lijh@qdu.edu.cn

Keywords: Test Automation, Reproduce Bug, Debug, Docker, Service.

Abstract. Test Automation is widely used. Since there are some differences between the environment of test automation and development, developers sometimes can't reproduce bugs in their local environment based on bug reports. To solve the issue, this paper introduces an approach that integrates the environment of test automation and development. The solution uses Docker to launch service of integrated environment. Compared with solution based on cloning virtual machine, the solution has three main advantages. It helps developers to reproduce bug and debug, fix bug and verify the fix; it launches an integrated environment in a short time; and it takes much less resources. The solution is valuable for both test automation and development.

Introduction

Over the past years, tools that help programmers quickly create applications with graphical user interfaces have dramatically improved programmers' productivity and the usability of applications. Meantime the complexity of the software is getting increased. That has increased the pressure on software testers, who are being asked to test more and more code in less and less time. Software testing becomes very crucial. Software testers need to improve their own productivity. Test automation is one way to achieve this [1, 2].

A test case used for automation testing includes data files and scripts for each step in one test case [3]. Automation test tools automatically create a database and records each steps' data into the database when executing the test scripts. When a test case failed, we can back up the related database which was created by the steps before the failed step, and then supply the database to developers to reproduce the bug.

When a new version of software is released, software tester will use test automation system to automatically run test cases to test the software. Then they will analyze the failed test cases and report incidents to developers. The environment of test automation usually only includes the tools for test automation and release version program, while there are development tools, debugging version program and source code in the development environment. Some incidents can only be reproduced in release version program. The complexity of environment forces developers to turn into detectives searching for an explanation of how the program could have arrived at the reported failure point [4] in their local development environment.

There are a number of works on reproducing bugs [5, 6] and software reproducibility [7, 8, 9, 10, 11]. And there is an existed solution based on virtual machine (VM) cloning to solve this problem. When software testers report a bug, they can submit a request to clone their current test automation VM, and then customize the cloned VM, and get related debugging version program and source code into the VM. They supply the new VM to developers. But as there are not tools for development in the new environment, test automation VM can't be used until the process of cloning is done, so this method is not convenient to both testers and developers. What's more, the process of cloning and getting debug version program and source code takes a very long time, and the new environment will take an army of resources.

This paper presents a new approach to solve current problems. The solution is based on Docker to automatically prepare environment that integrates both test automation and development's

environment including the tools of test automation and development, release version program, debugging version program and source code, and then based on developers' request to automatically launch service to them. It can reduce developers' effort required to setup environment for reproducing bug and debugging. Developers can use the integrated environment to reproduce bug, debug, fix bug and verify fixes. Furthermore, it uses Docker container to launch the service in an extremely short time, and it takes much less resources based on layer file system of Docker, and there are a lot of elastic resources based on developers' demand.

The rest of the paper is organized in the following way. Section 2 presents related theoretical foundations: Docker. Section 3 presents the content of using Docker for reproducing bug and debugging. Section 4 describes the summary.

Docker

Docker is an open source operation system level virtualization technology that builds on LXC container, Control Groups and Union file system. Details on Docker can be found at: http://docs.docker.com. Here this paper's focus is on introduction about implications it has for reproducing bug and debugging in complex environment. Docker provides many attractive features for reproducing bug and debugging research:

- Docker containers are lightweight, and it can be launched in sub-second times, while VM cloning takes a long time.
- Unlike a VM, the contents in a container are reverted to their original condition of the Docker image when the container is launched. This way can ensure a consistent environment [12].
- One of reasons Docker containers can be portable and lightweight is because of union file systems. Union file systems are file systems that operate by creating layers.
- For resource management in Docker, containers only take the resources they are actively consuming.
- Docker can use Dockerfile to automatically build Docker images. Dockerfile provide a simple script that defines exactly how to build up Docker images.

Using Docker for Reproducing Bug and Debugging

The existed solution based on VM cloning for reproducing bug and debugging has some defects as mentioned in the introduction section. The following is a solution we have developed, which uses Docker to provide the services of reproducing bug and debugging.

Integrated Environment. There are some differences between the environment of test automation and development. The development environment includes source code, debugging version software and development tools. While there are test automation tools and release version software in the environment of test automation. The final environment we supply to developers for reproducing bug and debugging integrates both the environment of test automation and development.

Service Form. As a multitude of software are developed on Windows platform, and test automation tools are based on Windows platform too. Consequently our services are launched to developers in the form of Windows VMs. However, Docker containers can run only Linux programs. Thus we can use KVM (Kernel-based Virtual Machine) to run Windows VM.

Function of Switching between Debugging and Release Version Environment. In order to use test automation tools to reproduce bugs in debugging version software, in the first place, we need to update the scripts of test cases to adapt debugging version software. There is one more point, we need an extra script to support the function of switching between release version software and debugging version software.

Support more than one code line of software. To support more than one code line of software, each code line of software corresponds with a Linux VM. These Linux VMs are in the same physical host, and our services for different code line are managed in the corresponding Linux VM. We can

easily manage the environment in the way of taking snapshots after finishing making the integrated environment and reverting to the initial state when there are issues in Linux VMs [13].

The Workflow of Using Docker to Reproduce Bug and Debug. The workflow of using Docker to reproduce bug and debug includes the workflow of automatically making the integrated environment and automatically launching services to developers.

The workflow of making integrated environment is listed as below:

- Prepare Linux template VM. Firstly, install Docker and pull a KVM Docker image from Docker Hub. Secondly, prepare a Windows image file which has been installed related test automation tools and development tools.
- When the latest release version software rollout, we clone the Linux template VM.
- Customize the cloned Linux VM after the process of cloning is done.
- Then login the cloned Linux VM, run the KVM Docker container to boot up Windows VM.
- After Windows VM boots up, install the corresponding release version software.
- Then get related source code and compile to generate debugging version software.
- Stop and remove KVM Docker container.
- Run another KVM container to shrink the Windows image file.
- After the integrated environment is done, use the Dockerfile to build the final Docker image.

The main workflow is summarized as in Fig. 1.

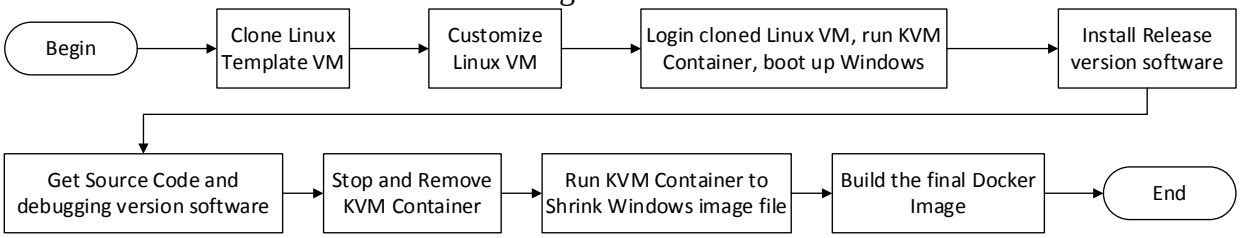

Fig. 1 The workflow of making environment.

After the final Docker image is ready, we base on the Docker image of integrated environment to launch Docker containers to provide services for developers.

The way of using integrated environment. When a software testers report a bug, they can run the final Windows Docker container to launch a Windows VM. And developers don't need to prepare environment for reproducing bug by themselves. The running environment of Windows VM is shown as Fig. 2.

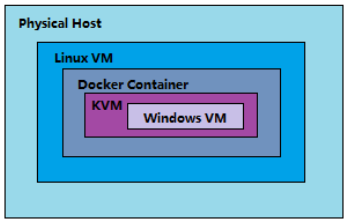

Fig. 2 The running environment of Windows VM

Developers can use the supplied service to reproduce regular bugs in debugging version software, and use release version software to reproduce the bugs which can only be reproduced in release version software. They can also verify their fixes after reproducing bug, debugging and fixing bugs.

Result Analysis. Compared with cloning VM, the way of running a Docker container based on the final Docker image of integrated environment can launch service in half-second. While the method of VM cloning takes about 40 minutes.

What' more, in contrast to VM cloning, we can use less resources to launch more Windows VMs. For instance, there is a host, which has 32 CPUs × 2.127GHz, 192GB memory and 2TB storage. Each Windows VM has 100GB storage, 4GB memory and 1 CPU × 2.127GHz. For the solution based on Docker, each Linux has 300 GB storage, 65 GB memory and 16 CPUs × 2.127GHz, and 15 services could run in each Linux VM. There are two Linux VMs corresponding two code lines except for the

Linux template VM. So the result comparison of total service count in the same host is shown in the Table 2.

Table 2 Result comparison of total service count

Solution	Total Service Count
Based on VM Cloning	17
Based on Docker	30

Summary

The method of this paper can supply developers the services to reproduce bug and debug, and can reduce developers' effort required to setup environment. In contrast to the existed solution based on VM cloning, the service supplied integrates both test automation and development environment, and it take much less time to prepare environment. What's more, it takes much less resources.

References

[1]. Bret Pettichord. Seven Steps to Test Automation Success. http://www.pettichord.com (Revised version of a paper presented at STAR West, San Jose, Nov 1999).

[2]. G. Mohan Doss Gandhi, Anitha S. Pillai. Challenges in GUI Test Automation. International Journal of Computer Theory and Engineering Vol. 6 (2014) No. 2.

[3]. Eun Ha Kim, Jong Chae Na, Seok Moon Ryoo. Implementing an Effective Test Automation Framework. Computer Software and Applications Conference. 2009.

[4]. Cristian Zamfir, George Candea, Execution Synthesis: A Technique for Automated Software Debugging. In ACM EuroSys European Conf. on Computer Systems. 2010.

[5]. Jeff Huang, Charles Zhang, Julian Dolby. CLAP: Recording Local Executions to Reproduce Concurrency Failures. In Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI '13.

[6]. M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar, I. Neamtiu. Finding and Reproducing Heisenbugs in Concurrent Programs. In OSDI, 2008.

[7]. P. Guo, D. Engler. CDE: Using System Call Interposition to Automatically Create Portable Software Packages. Proceedings of the USENIX Annual Technical Conference. 2011.

[8]. C. Collbery, T. Proebsting, G. Moraila, A. Shankaran, Z. Shi, A. M. Warren. Measuring Reproducibility in Computer Systems Research, 2014. Department of Computer Science. University of Arizona, Tech. Rep. 2014.

[9]. Peng, R.D, Reproducible research in computational science. Science. 334. Dec. 2011.

[10]. Carl Boettiger. An introduction to Docker for reproducible research. ACM SIGOPS Oper. Syst. Rev. 49(1). 2015, p. 71-79.

[11]. Chamberlain, R., Invenshure, L., and Schommer, J. Using Docker to Support Reproducible Research. Technical Report 1101910. 2014.

[12]. D.Merkel, Docker: lightweight linux containers for consistent development and deployment. Linux Journal. Vol. 2014 No. 239. 2014, p. 2.

[13]. Xiaoqiao Meng, Vasileios Pappas, Li Zhang, Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. IEEE INFOCOM 2010 proceedings.