On Construction of a Rapid Virtual Development Environment for VMware Ecosystem

Chen-Kun Tsung[1*], Xin-Ting Zhang [1], Ming-Cheng Tsai [2], Chih-Wei Wu [2]

ABSTRACT

When developing cloud services, the engineers would rapidly maintain the development environments, e.g., the Platform as a Service, for creating, modifying, and destroying environments. VMware provides some services to realize the about necessary. However, the major issues in considering the above services are the integration. The above services fit the necessary of the development environment, but the engineers would switch the working platform for various purposes. To increase the service integration, we have to develop the core service and encapsulate it as the core service modular, so the applications could involve the modular to realize various purposes. In this work, we develop the Rapid Virtual Development Environment (RVDE) to realize the above necessary. The RVDE applies VMware REST APIs to the major action invocations; Node.js is considered to bridge the user commands and actions; Ansible Playbook is used to create the major action scripts. The engineers using the RVDE provide the necessary parameters for the target environments, and the RVDE will automatically create the development environments based on the given parameters, e.g. IP addresses, login information, and other system configurations. The RVDE provides easy and rapid use from the experiments, and the environment creation time is saved by 90% for the environment configuration.

Keywords: API integration, VMware, Cloud Computing, Platform as a Service

*Corresponding Author: Xin-Ting Zhang (E-mail: 3a817013@gm.student.ncut.edu.tw).
[1] Department of Computer Science and Information Engineering National Chin-Yi University of Technology, No.57, Sec. 2, Zhongshan Rd., Taiping Dist., Taichung 411030, Taiwan (R.O.C.)
[2] Mechanical and Mechatronics Systems Research Labs, Industrial Technology Research Institute, No. 195, Sec. 4, Chung Hsing Rd., Chutung, Hsinchu, 310401, Taiwan (R.O.C.)

I. INTRODUCTION

Virtual machines (VM) are useful in increasing host utilization, and the cost of maintaining servers would be reduced. Therefore, a lot of applications consider the VMs to be the infrastructure. For example, environment detection [1] [2] [4], illness prediction [3] [6], manufacturing [8] [9], scheduling [7] [10], and education [5]. The architecture of VMs is illustrated in **Figure 1**. The VMs share the host resource with the hypervisor. The single host may provide several VMs so that the host utilization could be increased in VMs.

However, the major issue of VMs is resource optimization. Hypervisor visualizes the hardware resources to support the virtual hardware for funning VMs, as shown in **Figure 1**. Therefore, each VM has a private and virtual hardware profile, and resource virtualization increases hardware utilization. However, the data redundancy is not optimized by the hypervisor. For example, the VMs with the operating system have the same system software, and they are duplicated in various VMs. The same so that the same files appear in several VMs, and the many spaces are used to save the same data.

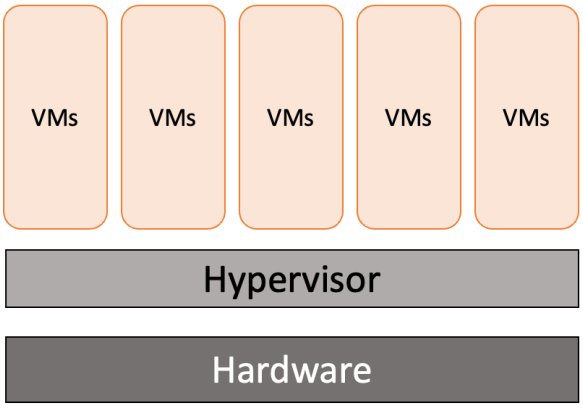

Figure 1. The architecture of VMs.

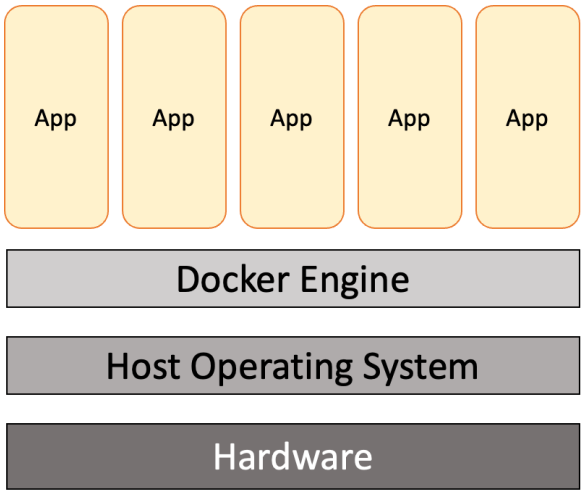

Figure 2. The architecture of Dockers.

Docker considers the shared memory architecture to enhance resource-sharing performance. As shown in **Figure 2**, the host has exactly one operating system. The docker engine runs based on the host operating system, while the docker applications share low-level resources, such as the common operating system service and the hardware instructions. Therefore, the common low-level resources are shared with docker applications, and only a few resources are required.

However, the major issue of docker is security [11] because the low-level resources are shared. Various docker applications use the same resource to execute different jobs, so the malicious applications would invalidly access other applications' data based on the shared resource architecture. Thus, access control in docker applications is a critical research topic [16].

On the other hand, the low-level resources are independent in the VMs, so the security issues of docker do not appear in VMs. Therefore, for the security issue, this work considers the VM to improve the efficiency of deploying VMs.

To resolve the above optimization issue of VMs, the dockers [12][13] and Kubernetes [14][15] are proposed to reduce the duplicated files. The docker architecture is drawn in **Figure 2**. The creation process includes three phases: 1) VM setup, 2) VM creation, and 3) VM configuration. In the VM setup phase, the specifications of VMs are determined and sent to VMware vSphere, such as CPU number, RAM size, and disk space. In the VM creation, VMware vSphere creates the VMs according to the received VM specification. After the second phase, the VMs are duplicated with the specified specification, but the software configuration is not changed, such as the user account and password. Therefore, the third phase, the VM configuration, modifies the configurations.

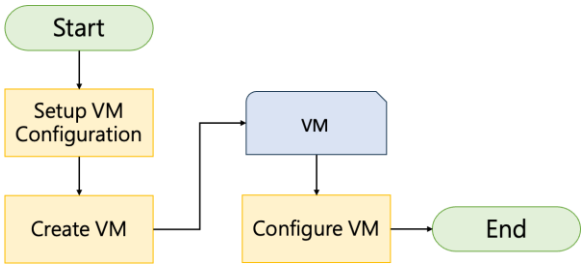

Figure 3. The flowchart of creating a VM.

From the implementation, the second phase process is executed by vSphere automatically, while users handle the first and third phase processes. The variance in the degree of VM customization increases the manpower requirement. In other words, creating VMs with various configurations require more manpower than that with single ones. Therefore, this work constructs an automatic VM deployment service, Rapid Virtual Development Environment (RVDE), to reduce manpower investigation and increase job efficiency.

From the experiment results, the proposed RVDE creates the specific VMs correctly. On the other hand, all VMs are automatically deployed by the proposed RVDE, so the manpower investigation is decreased. In other words, the feasibility and the efficiency of the proposed RVDE are confirmed.

Section II describes the proposed approach RVDE in the following content of this article to build up specific VMs. We apply some experiments to evaluate the efficiency and feasibility of the proposed RVDE, and the results are discussed in Section III. The conclusion and future works are illustrated in Section IV.

II. PROPOSED METHOD:

The software architecture of the proposed RVDE platform is illustrated in **Figure 4**. The RVDE platform is built on Windows 10. The major services include Node.js, WebSocket, Express, and Ansible. Node.js is used to communicate and integrate the entire service of the proposed RVDE; WebSocket is used to exchange the data between the proposed RVDE platform and the VMware, which includes the VMware REST API and VMware vSphere; Express is the integrated package for Node.js; Ansible is used to modify the system configuration of the target VMs.

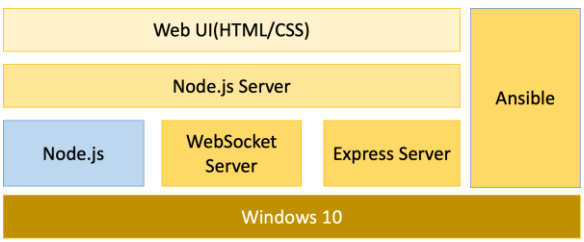

Figure 4. The service architecture of the proposed Rapid Virtual Development Environment platform.

The proposed RVDE platform consists of three processes: 1) the user provides commands, 2) the RVDE creates VMs, and 3) the RVDE modifies the system configurations of the created VMs. The architectures of the above processes are illustrated in **Figure 5**, **Figure 6**, and **Figure 7**, respectively. The details of each process are described in the following.

The user interaction diagram for the communication between the user command and the service script creation is drawn in **Figure 5**. The user on the left-hand side of **Figure 5** provides the system configurations of VMs via the user interface of the RVDE. The RVDE generates the Ansible playbook by Node.js, and the Ansible playbook is constructed in the YAML format. Then, the Ansible playbooks will be delivered to Ansible. Eventually, the system configurations will be sent to VMware REST API to do the actions.

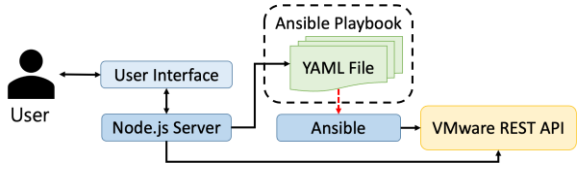

Figure 5. The user interaction diagram for the communication between the user command and the service script creation.

The details about the actions of the playbook generations are illustrated in **Figure 6**. Node.js in the RVDE consists of two major functions: the WebSocket Server and Express Server. The WebSocket Server provides the low-level process of the network access service, while the Express Server provides the necessary functions for Node.js. Express Server generates the Ansible playbook in YAML format, which describes the system configuration of VMs. Therefore, the actions of VM creation are divided into two jobs: 1) the VM creation, and 2) the VM modification, and the jobs are handled by Express Server. On the other hand, the RVDE invokes VMware REST API to create the VMs. Since the VM creation process only duplicates the VM by the specified VMs or VM templates. In other words, the Express Server sends the VM creation actions to the VMware REST API and the system configurations to Ansible.

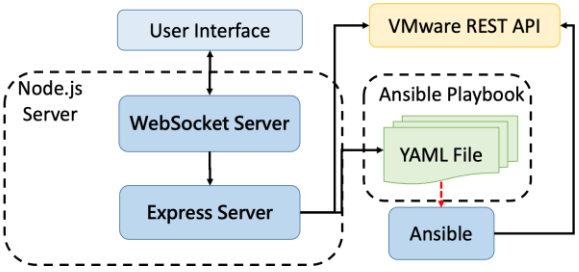

Figure 6. The architecture of creating the system configuration script is based on the user's command.

The actions between the Express Server in **Figure 5** and the VM creation actions are illustrated in **Figure 7**. VMware REST API makes the major process of creating VMs in the VMware vSphere. Since the resources of real hosts are virtualized by vSphere, all processes are well monitored under vSphere. After receiving the VM creation missions, VMware REST API duplicates the VM by the specified VMs or VM templates. The duplications will be assigned the resource, and the records will be saved in the database. All the above VM creation processes are processed by VMware REST APIs. The creation progress will be returned, and the user can monitor the current status of the VM creation.

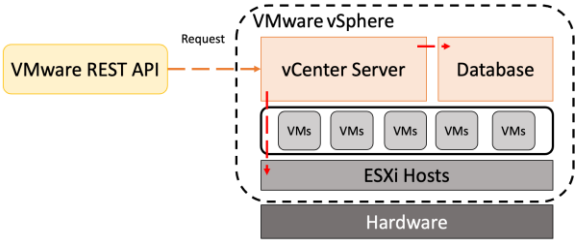

Figure 7. The architecture of service communication between the VMware REST API and the hosts.

III. EXPERIMENTAL ANALYSIS

The usage of the proposed RVDE consists of two sides: 1) the server side, and 2) the client side. The VMs are created on the server side while the commands are received from users on the client side. We consider the server farm with five hosts for the server side, while two workstations are used to be the client side as shown in **Figure 8** and **Figure 9**, respectively. The software and hardware specifications are listed in **Table 1** and **Table 2** for the server and client sides, respectively.

On the server side, the host resource is virtualized by VMware ESXi 6, and 10Gb Ethernet connects the storage

and the server farm. All VMs are deployed in the storage rather than the local storage. The vCenter handles the VM maintenance on the server side. Therefore, the server farm is controlled by vCenter because of the virtualization maintenance, such as the high availability, power balancing, and load balancing.

Table 1. The hardware and software specification of the test server farm.

	Specifications
# of server	5
CPU (per host)	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
RAM	192 GB
Storage	MSA 2050 9 TB
Hypervisor	VMware ESXi, 6.0.0

Figure 8. The experiment server farm is used to evaluate the performance of the proposed RVDE.

Two Windows-based workstations handle the process on the client side. The Linux-based platform provides higher performance with lower hardware resource requirements. However, the target audience of the RVDE is basic IT engineers, so the Windows-based platforms are more friendly than the Linux-based platforms. The specification of the workstations is similar to the personal computer, but the workstations provide higher stability and redundancy than personal computers. Therefore, the workstations are more appropriate than personal computers.

Table 2. The hardware and software specifications of the experiment client workstations.

Specifications	
# of server	2
CPU (per host)	Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz 3.00 GHz
RAM	64 GB
Storage	CT500MX500SSD 500 GB

Figure 9. The experiment client workstations which are used to access the proposed RVDE and provide the commands.

In the experiment, we consider two batches with various IP addresses to evaluate the correctness of VM creation. The specifications of the two batches are illustrated in **Figure 10** and **Figure 11**, respectively, while the created VM results are drawn in **Figure 12** and **Figure 13**. The VMs' names and the IP addresses are the major differences in the two experiments. For convince, we consider two VM names to identify the VMs. The VMs are connected to the network when VMs are created, so the IP addresses should be different.

Batch 1

Number of virtual machines : 3
Name of the virtual machines : test
Folder location : ST_test_20220331
Password : •••
Cluster name : Research
Database : Datastore 23
Number of CPU cores : 2
Disk space (GB) : 1024
Memory space (MB) : 2048
Initial IP address : 192.168.0.91
IP address settings : Increment the last segment of the IP address

Figure 10. The VM specification of the first test.

Batch 2

Number of virtual machines : 3
Name of the virtual machines : qwe
Folder location : ST_test_20220331
Password : •••
Cluster name : Research
Database : Datastore 23
Number of CPU cores : 2
Disk space (GB) : 1024
Memory space (MB) : 2048
Initial IP address : 192.168.0.944
IP address settings : Increment the last segment of the IP address

Figure 11. The VM specification of the second test.

Batch 1

Template name :	(TEST)[SRC]Ubuntu_Server_20.04_8G_1T
Datacenter :	Datacenter
Number of virtual machines :	3
Name of the virtual machines :	test
Folder location :	ST_test_20220331
Password :	123
Database :	Datastore 23
Cluster name :	Research
Number of CPU cores :	2
Disk space (GB) :	1024
Memory space (MB) :	2048
Initial IP address :	192.168.0.91
IP address settings :	Increment the last segment of the IP address

Figure 12. The result of creating VMs according to the specification in the first test.

Batch 2

Template name :	(TEST)[SRC]Ubuntu_Server_20.04_8G_1T
Datacenter :	Datacenter
Number of virtual machines :	3
Name of the virtual machines :	qwe
Folder location :	ST_test_20220331
Password :	123
Database :	Datastore 23
Cluster name :	Research
Number of CPU cores :	2
Disk space (GB) :	1024
Memory space (MB) :	2048
Initial IP address :	192.168.0.944
IP address settings :	Increment the last segment of the IP address

Figure 13. The result of creating VMs according to the specification in the second test.

Figure 14. The specification of created VM with the name qwe_2.

The experiment results illustrated in **Figure 12** and **Figure 13** are captured by the RVDE logs. The specifications of all created VMs are the same as that provided by the user. Therefore, the proposed RVDE correctly creates the VMs according to the specifications provided by the user. Moreover, to evaluate the actual specification, we capture the specification of created VM named qwe_2 that is detected by vSphere, and the result is illustrated in **Figure 14**. The hardware specification of the VM qwe_2 is the same as that provided by the user, including the CPU number, memory size, disk space, and IP address. In other words, the proposed RVDE provides correct and automatic VM creation.

IV. CONCLUSIONS

In this paper, a rapid VM creation service, which is named RVDE (Rapid Virtual Development Environment) is proposed to save manpower to set up the VMs and configure the systems. To create a large number of customized VMs, the information and technology engineers should set up VMs by the specifications and input the VM specifications to the hypervisor, e.g., VMware vSphere. Then, the hypervisor creates the VMs according to the VM specifications. After the VMs are created, the information and technology engineers have to enter each VM to set up the system configuration. For example, the user account, login password, and IP address. The proposed RVDE decreases the time investigated by the information and technology engineers. Moreover, the proposed RVDE provides the batch process, and in other words, the information and technology engineers only require preparing the system specification for the RVDE, and the RVDE will create the VMs automatically. The manpower is released by the RVDE.

The RVDE could be applied to deploy a large number of VMs. For example, the education needs to create several VMs for the implementation topics in a class, and the RVDE could help in this scenario. Adjusting the scale of

cloud service. The high-performance computing needs huge computing power to finish the jobs. Deploying physical devices to provide computing power is a straightforward solution. However, computer power is not always necessary, and it takes place only when the jobs are coming. When the jobs arrive, the engineers could use the RVDE to create VMs rapidly to increase the scale of the computing power. So, the response time of the cloud service would be reduced by the RVDE.

The proposed RVDE creates customized VMs rapidly, but managing the ecosystem of cloud service still needs to consider more services to increase the service completeness. For example, the service destroy is a critical service. When the mission is completed, the VMs should be destroyed to release the hardware resources. On the other hand, the system service should be monitored online to guarantee the performance of each VM. Therefore, in the future, we will consider the above two issues to increase the completeness of the RVDE.

ACKNOWLEDGMENTS

This study is supported in part by the National Science Council of the Republic of China under contract numbers MOST 109-2221-E-167 -030 -MY3 and NSTC 112-2221-E-167-035 -.

REFERENCES

[1] Yang, C. T., Chen, S. T., Liu, J. C., Liu, R. H., & Chang, C. L. (2020). On construction of an energy monitoring service using big data technology for the smart campus. Cluster Computing, 23, 265-288.

[2] Kristiani, E., Yang, C. T., & Huang, C. Y. (2020). iSEC: An optimized deep learning model for image classification on edge computing. IEEE Access, 8, 27267-27276.

[3] Yang, C. T., Chen, Y. A., Chan, Y. W., Lee, C. L., Tsan, Y. T., Chan, W. C., & Liu, P. Y. (2020). Influenza-like illness prediction using a long short-term memory deep learning model with multiple open data sources. The Journal of Supercomputing, 76, 9303-9329.

[4] Kristiani, E., Yang, C. T., Huang, C. Y., Wang, Y. T., & Ko, P. C. (2021). The implementation of a cloud-edge computing architecture using OpenStack and Kubernetes for air quality monitoring application. Mobile Networks and Applications, 26, 1070-1092.

[5] Kristiani, E., Yang, C. T., Huang, C. Y., Ko, P. C., & Fathoni, H. (2020). On construction of sensors, edge, and cloud (ISEC) framework for smart system integration and applications. IEEE Internet of Things Journal, 8(1), 309-319.

[6] Tsung, C. K., & Tso, Y. A. (2022). Recognizing Edge-Based Diseases of Vocal Cords by Using Convolutional Neural Networks. IEEE Access, 10, 120383-120397.

[7] Wu, Q., Xie, N., Zheng, S., & Bernard, A. (2022). Online order scheduling of multi 3D printing tasks based on the additive manufacturing cloud platform. Journal of Manufacturing Systems, 63, 23-34.

[8] Haghnegahdar, L., Joshi, S. S., & Dahotre, N. B. (2022). From IoT-based cloud manufacturing approach to intelligent additive manufacturing: Industrial Internet of Things—An overview. The International Journal of Advanced Manufacturing Technology, 1-18.

[9] Ghafari, R., Kabutarkhani, F. H., & Mansouri, N. (2022). Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. Cluster Computing, 25(2), 1035-1093.

[10] Zhou, D., Xue, X., & Zhou, Z. (2022). SLE2: The improved social learning evolution model of cloud manufacturing service ecosystem. IEEE Transactions on Industrial Informatics, 18(12), 9017-9026.

[11] Patra, M. K., Kumari, A., Sahoo, B., & Turuk, A. K. (2022, December). Docker Security: Threat Model and Best Practices to Secure a Docker Container. In 2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC) (pp. 1-6). IEEE.

[12] Kim, B. S., Lee, S. H., Lee, Y. R., Park, Y. H., & Jeong, J. (2022). Design and implementation of cloud docker application architecture based on machine learning in container management for smart manufacturing. Applied Sciences, 12(13), 6737.

[13] Halenárová, L., Halenár, I., & Tanuška, P. (2022, September). Digital Twin proposal using the Matlab-Stateflow model and Docker containers. In 2022 Cybernetics & Informatics (K&I) (pp. 1-6). IEEE.

[14] Liu, J. C., Hsu, C. H., Zhang, J. H., Kristiani, E., & Yang, C. T. (2023). An event-based data processing system using Kafka container cluster on Kubernetes environment. Neural Computing and Applications, 1-18.

[15] Cha, J. H., Jeong, H. G., Han, S. W., Kim, D. C., Oh, J. H., Hwang, S. H., & Park, B. J. (2023, July). Development of MLOps Platform Based on Power Source Analysis for Considering Manufacturing Environment Changes in Real-Time Processes. In International Conference on Human-Computer Interaction (pp. 224-236). Cham: Springer Nature Switzerland.

[16] Guo, J., Xiao, J., Liu, Z., Cheng, Z., Liu, X., & Qiang, Y. (2022, July). Security Access Control of Docker Process Based on Trust. In International Conference on Artificial Intelligence and Security (pp. 557-570). Cham: Springer International Publishing.

Chen-Kun Tsung received the B.Sc. and M.Sc. degrees in computer science and information engineering from Dayeh University, Changhua, Taiwan, in 2004 and 2006, respectively, and the Ph.D. degree in computer science and information engineering from National Chung Cheng University, Chiayi, Taiwan, in July 2014. He is an Associate Professor of Computer Science and Information Engineering with National Chin-Yi University of Technology, Taichung, Taiwan. His current research interests are in cloud computing, big data, Web-based applications, and combinatorial optimization.

Xin-Ting Zhang received a B.Sc. degree in Computer Science and Information Engineering with National Chin-Yi University of Technology, Taichung, Taiwan, in 2023. His research interest is Cloud Computing.

Ming-Cheng Tsai received a M.Sc. degree in Department of Mechanical Engineering with National Taiwan University of Science and Technology, Taipei, Taiwan, in 2016. His research interest is online measurement and mechanical design.

Chih-Wei Wu received a M.Sc. degree in Introduction of Department of Mechanical and Electro-Mechanical Engineering with National Sun Yat-sen University, Kaohsiung, Taiwan, in 2007. His research interest is six-bar human-like robotic.