ADV MATH SCI JOURNAL
Advances in Mathematics: Scientific Journal **9** (2020), no.7, 5065–5073
ISSN: 1857-8365 (printed); 1857-8438 (electronic)
https://doi.org/10.37418/amsj.9.7.72 Spec. Iss. on AMABDA-2020

A STUDY, ANALYSIS AND DEEP DIVE ON DOCKER SECURITY VULNERABILITIES AND THEIR PERFORMANCE ISSUE

AAKRITI SHARMA[1] AND RASHID HUSSAIN

ABSTRACT. Virtualization consists of the implementation of resources via software, although you can use additional resources at the firmware level or hardware it is a technique that can be applied in various contexts. The implementation of containers by Docker simplifying them and getting an easy implementation has allowed its use to be extended widely. However, information and awareness about a Correct and safe implementation of it has not gone hand in hand. Through this work it has been detected that Docker requires multiple approaches regarding the security that apply to each phase of Docker implementation. However, such information is the most widely dispersed and scarce compared to the amount of information that can currently be found about Docker. On the other hand, by observing in detail the different stages that make up the deploying an application with Docker safely, they have found enough differences in usability and capacity in Free tools available. In the search and creation stage of our image has been easy to find websites and applications that analyze them and provide a report with vulnerabilities and recommendations.

1. INTRODUCTION

In general terms, virtualization consists of the implementation of resources via software, although you can use additional resources at the firmware level or hardware it is a technique that can be applied in various contexts. So by in

[1]*corresponding author*

2010 *Mathematics Subject Classification.* 91G20, 97I80.

Key words and phrases. component, formatting, style, styling, insert.

the operating systems, for example, mechanisms have been used for decades we have virtual memory management, which allows a process to use an address space larger than that offered by the hardware where it runs. In the context of this work, we will refer to virtualization of resources at a level of major abstraction: virtual machines (VMs) and containers. A review of hypervisors and containers is presented in the following sections. Does These two technologies have often been presented as competitors, if We attend to the number of publications dedicated to comparing their performance [1] however, in practice they should be considered complementary, taking into account the widespread use of deploying container platforms on Virtual machines.

1.1. **Hypervisors.** A hypervisor or VMM (Virtual Machine Monitor) is a component implementing normally in software that allows to create and execute machine instances virtual, through hardware virtualization, and possibly other techniques additional, such as dynamic code translation. A virtual machine created on a hypervisor you can run a complete operating system, include-do the kernel, so it is possible to run virtual machines with different systems operational issues (for example, Windows and Linux) on the same hardware. Since Goldberg [2], hypervisors have been classified into two types based on environment where they run:
Type I. It runs directly on the CPU of the host computer. It is also nominate native hypervisors or bare-metal hypervisors. The hypervisor is a micron-Cleo is responsible for core operations, such as planning tasks or processes in the CPU (scheduling) , interrupt handling and management of memory, and delegates the rest of the operations to the cores of the virtual machines.
Type II. They run on the operating system of the host computer. They usually buy get two parts: one that runs as a module of the system core operating host, and another that runs as user-level processes in the system host operating theme. Task planning is executed at the core of the system. Host theme. In the literature the terms bare metal hypervisor and hosted frequently appear hypervisor as synonyms of type 1 and 2 hypervisors respectively.

1.2. **Application containers.** The pattern of habitual use of the container technologies to which we have referred to above in providing an execution environment similar to that obtained NE with a virtual machine. That is, use a container as an alternative to the use of a VM executed on a hypervisor. In the case of

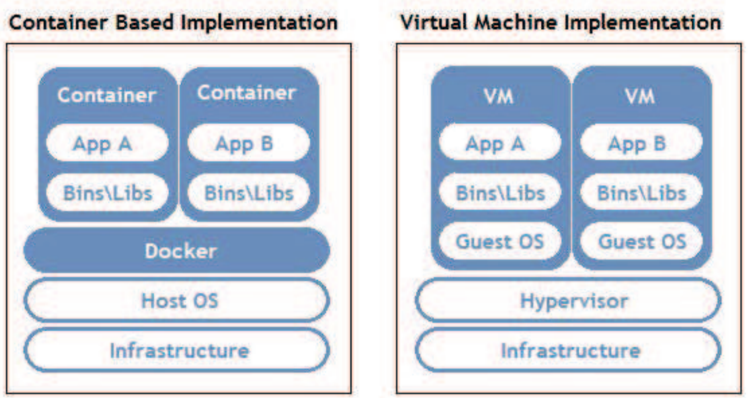

FIGURE 1. Docker VS Virtual Machine architectures

Linux, use case normal with LXC is to start the container running init or system, and the pro-usual stops: bash, sshd, etc. The most recent trend is, however, to use application containers. In this model, the recommended usage pattern is oriented to micro services: each container executes a group of applications or services determined in function of some policy (for example, segregating groups based on type of application, or of their consumers), instead of an execution environment complete to simulate a VM. It is important to note that in Linux, the main di-difference between an application container and a system container is your usage pattern: the core provides the same mechanisms for its creation in both cases.

The use of virtualization at the operating system level has come using for years to create what could be understood as "machines virtual light "(for example, Jails in FreeBSD, Zones in Solaris, and OpenVZ or LXC) platforms for the execution of application containers (such as Docker or Rkt from CoreOS) form a paradigm that introduces new innovations such as the use of structured images in layers, and facilities for the distribution of images. In parallel, this new model introduces new vulnerabilities, which must be offset by a set of mechanisms we are not very homogeneous, and often poorly documented. The use of application containers has experienced remarkable growth. in recent years, especially when compared to the growth of virtualization platforms based on hypervisors in the same period. This faith nome is due, among other causes, to the shorter starting times, less memory usage, and higher density of instances per container machine With respect to virtual machines.

1.3. **System Containers.** While the function of a hypervisor is, in some way, to partition of a machine, container-based technology creates systems of an instance of the operating system. Each partition maintains the appearance of being an independent machine, although they all share the same core of the OS[4,5]. This allows executing with a single instance of the system core. Operating theme multiple applications isolated from each other. The technique used in the containers has its origin in the system call root (2), present in UNIX since the seventies, and that allows a process change the root directory that was used to boot the system to another direct-River. Jails [6], implemented in FreeBSD at the end of the 1990s, employs a root-like technique to virtualize the file system of a set of applications (that is, modify the namespace of the file system visible from those applications), but adds similar measures to the spaces of names of processes and network resources. In 2004, a similar functionality, called "zones" [7] appeared on Solaris10. Solaris 10 zones improved partition isolation using are source aggregation scheme (resource pools), which allows allocate a set of resources (for example, memory or CPUs) so that partition use it exclusively. This prevents the workload of a partition monopolize the resources of the machine. Although in Linux, the functionality to support containers has had several implementations, the first project that obtained an appreciable diffusion was OpenZV.

2. The Dockers ecosystem

Following the popularity of Dockers, a considerable number of projects have appeared related coughs, which introduce novelties in technology, or provide alternative implementations similarly, many projects disappear with the same speed as new ones appear. It is possible that the speed at which they produce these new developments and the ever-changing landscape of the architectures of existing products be responsible for the confusing vocabulary Used in these technologies, where in the absence of clear definitions, meta-foraâĂŹs as a life cycle and ecosystem.A major issue of this ecosystem, which is mainly driven by open source projects, is the lack of clear functional demarcation of different components. In fact, capabilities and scopes of existing projects overlap. Similarly, we found Terminology in literature to be inconsistent. âĂIJIn particular, the name Docker has been used with different meanings. When it appeared in 2013, the situation was well defined: it was a pro-Free software project implemented in the Go language, and developed by a pro-PaaS seer called dot Cloud. Currently, Docker is a company (Docker Inc., a registered trademark) that markets various products, services, and promotions na several free software projects related to container technology Dores In the rest of the work, we will use the term Docker informally to re-We rely on a container management platform composed of three components.

- Engine, daemon
- Docker client
- Docker registry

3. Alternatives to Docker

Previously called Rocket is an alternate container platform-Docker, or more specifically, a container runtime, developed by Core OS. Unlike Docker, Rkt seems to follow the UNIX tradition of using tools. While you specialize in doing a good thing and that can be combined to realizer more complex operations. According to its creators, the tools for Upload images, install and run containers must be well integrated, but be independent; image distribution must support various protocol solos, and deployments in private environments should not require access to registry. Core OS provides, anyway, a registry for images of both rkt as from Docker.

4. Docker Security

Docker containers leverage the ability of the Linux kernel to create isolated environments that share the kernel with the host host. One of the characteristics of the kernel that is used are the "capabilities". I know divide the root capabilities so that each process can be granted capabilities that really requires. There are capacities for almost all the areas where root privileges are needed. This way you get avoid overuse of root users and use a permissions policy minimum that provides greater security to the system. If an intruder gets climbing to root inside the container will be more complicated to perform actions harmful or get scaled to the host. In most cases, Docker does not need root privileges and could be reduce capacities by denying among others:

- "mount" operations
- Access to raw sockets (to prevent impersonation attacks)
- Access to file system operations such as creating new device nodes, modify the owner of the files or alter the attributes
- Module loading

By default, Docker uses a white list. Deny all capabilities except those specified and supports the removal and insertion of new capabilities to list. It is important to review the ones that are really required by our process to establish the most secure configuration since the initially granted to containers create incomplete insulation.

5. Docker Images Security

To create our image we will use a base image of the stored in the Docker Hub. It is important not to trust any image found on the internet or in any repository. Docker sponsors a team dedicated exclusively to review and publish content in the official repositories. There are official repositories for more use cases commons that provide clear documentation and promote good practices so it is a good strategy to start from one of them. For the project, a base image for Wildfly servers is sought with the Docker search command. Docker provides the "Content trust" functionality, this option forces the image tags to be signed and verified in the customer side It is disabled by default so we must run the command:

<< export DOCKER_CONTENT_TRUST = 1 >>

We note that it is not signed so the option should be used Âń–Disable-content-trustÂż. It is always better to have it enabled by default and use the deactivation parameter if necessary for us to be aware that it is not signed and we should check it better. In any case and more in cases where official images are not used or not is signed, the images should be reviewed to avoid Introduction of harmful software in our container.

6. Comparative Analysis of docker Security

Container clusters and orchestration The micro service-oriented container use model often implies the need to maintain a considerable number of container clusters. In Many cases, these clusters may be geographically dispersed, and should meet minimum availability and fault tolerance requirements. The complex-The maintenance of such an infrastructure has caused the appearance of some software components that facilitate integration and container management. In general, the component that provides several or all functionalities needs SaraiâĂŹs are known as container orchestrators .The terminology, as usual in this field, is elastic enough so that you can't find two equivalent definitions of orches Tador; we will use as a basis the list of functionalities, because it excludes the possibility of leaving something out Thus, in a broad sense, the orchestrator is responsible for:

(1) Cluster status management and planning. It is required to maintain a life global status of the cluster state to be able to plan workloads depending on the free resources of the cluster and the schedule. This, the planfication of work queues of a conventional environment, but taking into the complexity of the container cluster counts.
(2) Provide high availability and fault tolerance: mechanisms for biliary fault detection and avoid single points of failures, balancing ofload.
(3) Security Integration of utilities to verify the integrity of the imagines, identity management and access management services, etc.
(4) Simplify network management: dynamic address and port management, communication between containers located in different machines through tunnels.
(5) Enable service discovery. In the model traditional services are associated in a relatively static way to direct ports and ports that are resolved by some combination of DNS, LDAP and HTTP URIs. The container model, the association between Service names and addresses cannot be persistent, among other things. Tries for the possibility of auto scaling (creation of dynamics of new instance.
(6) Make continuous deployment possible. When the CI / CD model is used, the Orchestration platform can provide mechanisms to integrate the Tool management like Jenkins.
(7) Monitoring of both the infrastructure where the containers are executed Dores like their activity.

(8) Docker's native container orchestration platform is Docker Swarm. The one that is receiving the most attention, however, is Kubernetes , who developed initially Google, and is currently a free software project.
(9) As a sequence of success of Kubernetes, Docker Inc. offers it as an alternative to Docker Swarm in many of its products, although not in the CE version for Linux.

Conclusion

After scratching the surface of the techniques used in the containers a little we have been able to verify that some of the ideas that we had at the beginning of the work were not entirely correct, or were directly so wrong. The idea of comparing a Linux container with a VM, in depending on its performance, density per host, and in particular in matters of security, as reproduced in so many publications does not seem so right. At the end of the work, our opinion is that VMs and containers of applications are different species, and therefore any comparison has little meaning: the containers are more or less confined process groups, and the safety comparison should be made with respect to processes that do not run in a container. From this point of view, the containers in Linux they bring substantial improvements in the security plane.

References

[1] C. Boettiger: *An introduction to Docker for reproducible research*, ACM SIGOPS Operating Systems Review, **49**(1) (2015), 71-79.

[2] W. Felter, A. Ferreira, R. Rajamony, J. Rubio: *An updated performance comparison of virtual machines and linux containers*, IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, 171-172. doi: 10.1109/ISPASS.2015.7095802

[3] A.S. Harji, P.A. Buhr, T. Brecht: *Our troubles with Linux Kernel upgrades and why you should care*, ACM SIGOPS Operating Systems Review, **47**(2) (2013), 66-72.

[4] M.J. Scheepers: *Virtualization and containerization of application infrastructure: A comparison*, 21st Twente Student Conference on IT June 23rd, 2014, Enschede, The Netherland.

[5] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, B.-J. Kim: *Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud,* Proccedings on Networking and Communication, 2014.

[6] W. VAN DER AALST, T. WEIJTERS, L. MARUSTER: *Workflow mining: Discovering process models from event logs*, IEEE Transactions on Knowledge and Data Engineering, **16**(9) (2004), 1128-1142.

[7] B. VARGHESE, L.T. SUBBA, L. THAI, A. BARKER: *Container-Based Cloud Virtual Machine Benchmarking*, IEEE International Conference on Cloud Engineering (IC2E), Berlin, 2016, 192-201. doi: 10.1109/IC2E.2016.28

[8] S. PRABU, V. BALAMURUGAN, K. VENGATESAN: *Design of cognitive image filters for suppression of noise level in medical images*, Measurement, bf141 (2019), 296-301.

[9] S. KESAVAN, E. SARAVANA KUMAR, A. KUMAR, K. VENGATESAN: *An investigation on adaptive HTTP media streaming Quality-of-Experience (QoE) and agility using cloud media services*, International Journal of Computers and Applications, 2019. https://doi.org/10.1080/1206212X.2019.1575034

[10] S. BELLON, R. KOSCHKE, G. ANTONIOL, J. KRINKE, E. MERLO: *Comparison and evaluation of clone detection tools*, IEEE Transactions on Software Engineering, **33**(9) (2007), 577–591.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINNERING,
SURESH GYAN VIHAR UNIVERSITY,
JAIPUR, INDIA.
E-mail address: aakritivashishtha@gmail.com

PROFESSOR,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINNERING,
SURESH GYAN VIHAR UNIVERSITY,
JAIPUR, INDIA.
E-mail address: rashid.hussain@mygyanvihar.com