Unix in the Cloud

Harry Trezise
Swinburne University of Technology
103060492@student.swin.edu.au

***Abstract*—This paper examines the current state of the Unix Operating System as applies to Cloud computing. This includes the technologies enabling containerisation and Virtual Machines in support of both Platform as a Service and Infrastructure as a Service. The advantages and disadvantages of Unix in comparison to other leading server Operating Systems are also discussed, as are applications of Cloud-based Unix technologies to common network services.**

I. Introduction

The ubiquity of Cloud-based services for provision of (common) Internet and application infrastructure lends importance to the Cloud optimisation and capacity for integration of the Operating Systems (OSes) utilised. Unix provides capabilities that lend both to composition of Cloud infrastructure as well as the provision of many Cloud-enabled networking services.

This paper begins with discussion of the advantages and disadvantages of implementing Unix for Cloud services as compared against other OSes. Windows serves as the primary point of comparison, due to it being Unix's closest competitor in terms of market share for server deployments (including approximately 20% share to Unix's 80% for public web servers [1]). For purposes of this paper, the term Unix is applied to Unix-derived (or 'Unix-like') systems including Linux (various distributions) and BSD (Berkeley Software Distribution, primarily FreeBSD), with distinctions made where specific/unique functionality is discussed. The subsequent section describes various technologies available to provide virtualisation, as is crucial to the Cloud, both for Infrastructure as a Service (IaaS, where users purchase virtual machines) and Platform as a Service (PaaS, where users purchase containers), including their benefits and drawbacks [2]. The paper concludes with a discussion of considerations for integration of common Unix-based networking services with the Cloud.

II. Advantages and Disadvantages of Unix in the Cloud

The underlying design of Unix-based Operating Systems lends well to Cloud applications. Through configurability of the kernel, it is possible for a user to include or exclude features as required for a given application (through re-compilation). Features may be system binaries or drivers and the ability to do so is owed to modularity (i.e., components are de-coupled and have a minimal number of dependencies). In doing so, an instance can be reduced to the minimal required functionality, thus benefitting both memory usage and processing power required. Such is critical where infrastructure is hosted subject to Cloud providers' usage-based payment models, ensuring maximum service is derived from cost.

Windows is inferior to Unix in this regard, with Server editions including a Graphical User Interface (GUI) which, while simplifying administration adds significant resource overhead, and is not strictly necessary in many cases. Furthermore, users cannot re-compile Windows from its supplied state. However, Microsoft does provide a GUI-less 'Windows Nano Server' product, which is suitable for certain cloud workloads [3].

Windows is also based on proprietary code (closed-source), whereas Unix is open-source software. The latter benefits from continuous assessment of the security posture of the kernel and system binaries by its development community, increasing the likelihood of early detection of vulnerabilities and decreasing the frequency of security issues present in release versions [4]. That said, Unix derivatives are certainly not immune to security vulnerabilities, a notable past example being the 2016 Linux stack vulnerability, which may have supported payload injection into an improperly secured TCP connection [5].

Compared to Windows, Unix also has greater fault tolerance and can receive software updates (including OS updates) without the downtime of a reboot owing to modularity. Both are characteristics highly beneficial for (desirably) 'always-on' systems.

Both Unix and Windows Server include virtualisation functionality in various forms, the details of which are subsequently discussed.

III. Virtualisation

A. Virtualisation in Cloud Computing

Virtualisation is a key enabler of Cloud computing and thus Cloud-based networking services, delivering the means for a provider to divide and supply to subscribers their "shared pool of configurable resources" [6].

A key purpose of virtualisation is to provide security by preventing access to resources between tenants, as well regulating access to the overarching control infrastructure (which manages the system and provides means for configuring secure interfaces as required).

There are two distinct forms of virtualisation applied in Cloud platforms.

The IaaS paradigm provides subscribers with Virtual Machines (VMs). VMs emulate an entire computer system, incorporating hardware as well as a full OS (kernel and system binaries) [7]. They allow the running of multiple virtual systems on a single host machine and support configuration to pull from distributed resources. Modern server OSes (where acting as hosts) provide a direct hardware interface specifically for virtualisation, allowing for performance not dissimilar to 'bare metal' (non-virtualised) systems.

PaaS, on the other hand, provides subscribers with containers. Containers are an alternate form of virtualisation, whereby processes and resources relating to an application or service are logically prevented from interacting with the wider system. Thus, containers support the separation of tenant

resources while removing the need for a wholly separate installation of a kernel and system binaries, provided those of the host system match the requirements of the service. The conceptual basis of a container is to restrict an application's access to the minimal level of resourcing required to effectively execute its function [8].

Notably, PaaS is built upon VM infrastructure as is made available to customers under IaaS. However, in this case the VMs are wholly managed by the Cloud provider, with interfaces supporting deployment, management, and orchestration [9]. Such saves configuration effort for developers/system owners who do not hold requirements for full access to VM-specific functionality.

B. Virtualisation Technologies

1) Containers

Unix has long provided functionality aimed at facilitating the grouping and separation of resources within a single host OS instance. This section identifies and describes some of the key tools providing or utilising Unix functionality to enable containerisation.

Unix's chroot (change root) command was introduced in 1979, allowing visibility of the filesystem to be restricted to a defined subset from the perspective of a specified process and its children [10]. Restriction does not extend to RAM and processing [11]. As such, chroot is not considered a security feature [12]. Furthermore, it provides a low level of assurance of effective file system restriction, even when correctly configured due to possibility of 'escape' [12].

FreeBSD introduced 'jails' in 1998, improving on chroot by virtualising jailed processes' access to the file system, user accounts, and network interfaces. Groups of each are stored in 'namespaces' within which resource access is confined, with the kernel maintaining synchronisation between virtualised copies and source data [10] [13]. For instance, a jail has its own users and root account, but these are stored in a namespace, separate to those of the host and thus have no access or privilege outside the jail environment [12]. Jails remain competitive with more recent container implementations (such as Docker, subsequently described) in terms of performance, particularly where a high performing network stack is desirable [14].

LXC (Linux Containers) are a more recently introduced feature of the Linux kernel, having similar functionality to jails with the addition of fine-grained performance controls. LXC utilises cgroups (control groups), supporting the creation of resource groupings at the kernel level, to which different priority levels may be ascribed [9]. For instance, it is possible to define an upper limit on permissible CPU time for a cgroup [15]. Furthermore, LXC can integrate with Mandatory Access Control (MAC) services such as SELinux (Security Enhanced Linux), thereby implementing a further level of separation by requiring that access to resources be undertaken only via a policy check [12]. Such may further reduce the likelihood of container 'breakout' (i.e., unauthorised access to the host).

Docker is a platform that extends LXC with a view to improving usability, thus reducing the resource and knowledge burden ascribed to container configuration and deployment [10]. The open access Docker Hub service provides downloads for various pre-built container images upon which applications/systems can be created [12].

Kubernetes is an open-source tool which provides a mechanism by which to 'orchestrate' (deploy and manage) large volumes of containers through use of configurable automation [16]. This has become critical for the functioning of distributed cloud-based applications whereby containers are utilised to run atomic modules that are created and destroyed as needed. Such allows an application to cater for user demand or the application's resource needs without a need for the continuous hoarding of resource capacity in the event of it being required. Docker Swarm is an alternative tool providing similar capabilities [17].

Notably, it is now possible to run Unix containers in Windows using Windows Subsystem for Linux (WSL). However, this service is more suited to testing and development than production deployment. There also now exists a native Windows container service (integrated with Docker) for workloads requiring integration with other Windows-enabled functionality [18].

2) Virtual Machines (VMs)

VMs allow an entire standalone system (or multiple) to be run using the underlying resources of a single host machine without sharing a kernel or having any access to the host filesystem (unless explicitly configured). Such a system comprises its own full OS, regardless of whether this matches with that of the host host [19]. A VM is entirely confined within its virtual memory allocation, with host machine communication provided as to a separate physical computer over a Local Area Network (LAN). In doing so, VMs support the deployment of virtual hosts with different operating systems or versions. For instance, a Unix host system can virtualise Windows and vice versa, providing versatility in how hardware resources are utilised for different workloads.

The hypervisor (such as FreeBSD's bhyve) is a key tool in VM-based virtualisation, as the means by which VMs are created and managed. 'Bare metal' hypervisors such as the Unix-based ESXi are minimally featured OSes designed solely for the purpose of virtualisation. Hypervisors utilising hardware-enabled virtualisation (e.g., VMWare ESXi and Microsoft Hyper-V) demonstrate significant performance advantages over emulation (involving the recreation of a processor architecture in software) as provided by systems such as QEMU [20].

Public Cloud platforms and systems such as OpenStack (often implemented in private Cloud systems) centralise management of host machines (termed as 'compute nodes') for use as a shared pool of resources on top of which to deploy VMs. For instance, [21] proposes use of FreeBSD systems as compute nodes upon which to host centrally managed VMs running, utilising bhyve. Platform users typically have no access to or control over the operation of the underlying host machines.

3) Container versus VM based Virtualisation

a) Efficiency

Efficient utilisation of computing resources saves costs for the customers of Cloud services. Containers can be deployed at a faster rate than VMs, lending well to applications in which a single operation is run on a container instance prior to it being destroyed [22] [23]. The container is re-generated only when its provided service is required. There is no need for a virtualised instance to remain continually running to be prepared for an event which may or may not happen. Such has

supported the creation of the current generation of modular, efficient Cloud native applications [19].

Containers are also more efficient from the perspective of size on disk, especially where specific features provided by an OS kernel that differs from that of the host are not required [19]. Opting for a VM in such circumstances may be viewed as needless duplication.

The higher performance of containers owing to the lack of hardware virtualisation overhead results in a performance gain which greatly benefits processing efficiency. Through utilisation of a single kernel, performance is closer to that of a physical machine than a VM [8]. For instance, the use of FreeBSD jails as opposed to VMs under certain conditions may have a performance benefit of over 50% [24].

From the perspective of a Cloud platform administrator, VMs support efficiency by allowing hosts to be of a single specification. Such reduces management overhead while customers remain free to use their desired/required OS on top of the base system.

b) Security

Owing to their separate kernel, VMs are further detached from a host in the event of an attack, thereby reducing the likelihood of malware leakage onto the host. Despite this, implementation of containers provides a degree of mitigation in the event of an attack by restricting access only to resources for which prior authorisation is held, particularly where MAC is also implemented [12]. It could be argued that this represents finer grained control than VMs where, despite being wholly sandboxed from the system, the entire VM may more easily be affected in the case of an attack owing to lesser control over processes. Use of VMs as the basis for a PaaS offering combines these security benefits from the perspective of a Cloud platform's owner.

Docker introduces a variety of additional potential attack vectors, including in the use of Docker Hub for the download of base images, broadening the attack surface beyond more manual methods of container configuration [22].

IV. Cloud Integration with Unix-Based Networking Services

In the context of Domain Name System (DNS) implementations, containers enable the creation and destruction of slave machines in accordance with demand. For large scale Cloud-based applications utilising containers (enabled via orchestration using Kubernetes or similar), it is crucial to provide a scalable DNS service able to continuously adapt in accordance with the application's dynamic nature. In such systems, DNS is critical for communication between the composite containers, thus directly impacting application responsiveness and user experience. While services such as Bind are suitable for a broad range of DNS applications, a specialised architecture is required for use in large-scale distributed application contexts, including features such as optimised memory management, whereby forwarder requests are handled as background tasks (with lower priority) in order to guarantee the greatest possible CPU time for local requests [16].

Cloud-based, container-enabled Dynamic DNS has applications for the Internet of Things (IoT), enabling communication with a vast array of devices via Uniform Resource Locators (URLs). Such supports more improves the ease of integration of services hosted on IoT nodes with other Cloud based platforms for various potential applications [25].

The Apache web server (httpd on Unix) is a prime candidate for greater integration with Cloud services, with infrastructure scalability highly beneficial for the efficient management of spikes in demand. [26] conceptualises a Cloud-based load balancing solution whereby worker servers (sitting behind a load balancer) can be automatically registered and unregistered while the site remains online. Such would enable the spawning and destruction of httpd containers to serve fluctuations in traffic. Similarly, wider implementation of Content Delivery Network (CDN) services for web content (making heavy use of containers) provides a means by which to counter Deliberate Denial of Service (DDoS) attacks by allowing scalability to both serve and analyse incoming traffic to determine its legitimacy [27].

V. Conclusion

This paper discussed the advantages and disadvantages of Unix as an OS for Cloud deployments. Subsequently, it explored some of the tools provided in Unix to enable IaaS and PaaS, constituting containers and VMs. The final section discussed applications of Cloud-based virtualisation to traditional Unix networking services, primarily DNS and HTTP servers.

Despite benefits for efficiency, containers will not replace VMs in the Cloud, but instead act as a complementary solution to enable efficiencies where suitable.

References

[1] W3Techs, "Usage statistics of operating systems for websites," W3Techs, 2 September 2023. [Online]. Available: https://w3techs.com/technologies/overview/operating_system. [Accessed 2 September 2023].

[2] O. Sefraoui, M. Aissaoui and M. Eleuldj, "OpenStack: toward an open-source solution for Cloud computing," *International Journal of Computer Applications,* vol. 55, no. 3, pp. 38-42, 2012.

[3] Microsoft, "Container Base Images," Microsoft, 17 March 2023. [Online]. Available: https://learn.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-base-images. [Accessed 3 September 2023].

[4] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities," in *Proc. Americas Conference on Information Systems (AMCIS)*, San Francisco, 2009.

[5] A. Quach, Z. Wang and Z. Qian, "Investigation of the 2016 Linux TCP stack vulnerability at scale," in *Proc. ACM on Measurement and Analysis of Computing Systems*, Online, 2019.

[6] P. Mell and T. Grance, "NIST Special Publication 800-145: The Definition of Cloud Computing," September 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf. [Accessed 29 August 2023].

[7] M. MacLeod, "Escaping from a virtualised environment: An evaluation of container breakout techniques," May 2021. [Online]. Available: https://supermairio.github.io/assets/pdfs/Dissertation.pdf. [Accessed 25 August 2023].

[8] Z. Li, "Comparison between common virtualization solutions: VMWare Workstation, Hyper-V and Docker," Greenville, 2021.

[9] S. Singh and N. Singh, "Containers & Docker: Emerging roles & future of cloud technology," in *Proc. International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, Karnataka, 2016.

[10] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing,* vol. 1, no. 3, pp. 81-84, 2014.

[11] P.-H. Kamp and R. N. M. Watson, "Jails: Confining the omnipotent root," in *Proc. 2nd International System Administration and Network Engineering Conference (SANE'00)*, Maasricht, 2000.

[12] M. Eder, “Hypervisor- vs. container-based virtualization,” in *Proc. Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, Munich, 2016.

[13] FreeBSD, “FreeBSD Handbook 4th Ed.,” 6 July 2023. [Online]. Available: https://download.freebsd.org/doc/en/books/handbook/handbook_en.pdf. [Accessed 25 August 2023].

[14] S. Weisz and M. Carabas, “FreeBSD virtualisation - improving block I/O compatibility in bhyve,” in *Proc. AsiaBSDCon 2019*, Tokyo, 2019.

[15] M. Kerrisk, “cgroups(7) - Linux manual page,” man7.org, 3 April 2023. [Online]. Available: https://man7.org/linux/man-pages/man7/cgroups.7.html. [Accessed 3 September 2023].

[16] H. Liu, S. Chen, Y. Bao, W. Yang, Y. Chen, W. Ding and H. Shan, “A high performance, scalable DNS service for very large scale container cloud platforms,” in *Middleware '18 Industry*, Rennes, 2018.

[17] A. Lingayat, R. R. Badre and A. K. Gupta, “Integration of Linux containers in OpenStack: An introspection,” *Indonesian Journal of Electrical Engineering and Computer Science,* vol. 12, no. 3, pp. 1094-1105, 2018.

[18] F. A. B. H. Ali, “A study of technology in firewall system,” in *Proc. 2011 IEEE Symposium on Business, Engineering and Industrial Applications (ISBEIA)*, Langkawi, 2011.

[19] C. K. Rudrabhatla, “Comparison of zero downtime based deployment techniques in public cloud infrastructure,” in *Proc. Fourth International Conference on IoT in Social, Analytics and Cloud (I-SMAC)*, Coimbatore, 2020.

[20] B. Djordjevic, V. Timcenko, N. Kraljevic and N. Macek, “File system performance comparison in full hardware virtualisation with ESXi, KVM, Hyper-V and Xen hypervisors,” *Advances in Electrical and Computing Engineering,* vol. 21, no. 1, pp. 11-20, 2021.

[21] A. Mitran, M.-E. Mihailescu, D. Mihai, S. Weisz, M. Carabas and N. Tapus, “FreeBSD as a compute node in OpenStack,” in *Proc. IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, 2020.

[22] T. Combe, A. Martin and R. Di Pietro, “To Docker or not to Docker: A security perspective,” *IEEE Cloud Computing,* vol. 3, no. 5, pp. 54-62, 2016.

[23] R. Dua, A. R. Raja and D. Kakadia, “Virtualization vs containerization to support PaaS,” in *Proc. 2014 IEEE Internation Conference on Cloud Engineering*, Boston, 2014.

[24] C. Antunes and R. Vardasca, “Performance of jails versus virtualisation for cloud computing solutions,” *Procedia Technology,* vol. 16, pp. 649-658, 2014.

[25] Z. Benomar, F. Longo, G. Merlino and A. Puliafito, “A Cloud-based and dynamic DNS approach to enable the web of things,” *IEEE Transactions on Network Science and Engineering,* vol. 9, no. 6, pp. 3968-3978, 2022.

[26] S. Heinzl and C. Metz, “Toward a cloud-ready dynamic load balancer based on the Apache web server,” in *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Hammamet, 2013.

[27] S. Pooja, Meeradevi, M. R. Mundada and P. Ramanathan, “The analysis of the content delivery network application deployment on Kubernetes platform,” *Webology,* vol. 19, no. 2, pp. 4190-4204, 2022.