A Pattern for a Virtual Machine Environment

MADIHA H. SYED, Florida Atlantic University
EDUARDO B. FERNANDEZ, Florida Atlantic University

A Virtual Machine Environment (VME) creates and manages Virtual Machines which are isolated units of execution with access to virtualized hardware resources. VMs are created by the VME upon user request. VMs are not new, but their significance has increased tremendously with the use of clouds as a means to offer sharing of resources and providing inexpensive, on-demand, isolated units of execution. A VME enables users to execute different types of operating systems and application stack in each VM. We present a pattern for a Virtual Machine Environment which describes its benefits and liabilities. Alternative solutions for portable execution environments have appeared, such as software containers, but still VMs continue to be the most versatile and flexible among the virtualization solutions because of their security and flexibility.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures - Patterns

General Terms: Design

Additional Key Words and Phrases: virtual machine, architecture patterns, security, virtualization, security patterns, software container

ACM Reference Format:

Syed, M.H. and Fernandez, E.B. 2016. A Pattern for a Virtual Machine Environment. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 23 (October 2016), 8 pages.

1. INTRODUCTION

Developing system programs, in particular operating systems, requires direct access to the hardware. For a company developing an operating system intended to run in several varieties of hardware, the only choice was to buy several hardware servers, which is expensive and limited this type of work only to big companies. When hardware meant mainframes, even large companies could not afford this development. This problem lead to Virtual Machine Operating systems which provided a set of replicas of the hardware architecture (Virtual Machines). VMs were first used in IBM mainframes so that designers could develop new system programs without crashing the whole shared system; these were implemented as the VM/360 and VM/370 architectures. We presented a pattern for VM Operating Systems in (Fernandez and Sorgente 2005) and (Fernandez 2013a). Later, VMs were used to share expensive hardware, such as web servers, which hosted web sites isolated from other sites. Now VMs have become the fundamental execution units (instances) of cloud systems and for a while they were the only unit of execution in this type of systems. Recently, containers have appeared and they can provide basically the same functions but using a fixed execution environment (Syed and Fernandez 2015); they trade off flexibility and security for speed. A Software Container provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers. Given the continuing value of VMs it is worthwhile to describe the environment used for their creation and management.

As shown in Figure 1, VMs and Containers are elements of cloud ecosystems (Fernandez et al. 2016b). An ecosystem is a collection of software systems, which are developed and co-evolve in the same environment. The Cloud Reference Architecture (RA) is the main pattern (hub) that defines this particular ecosystem. A Reference Architecture (RA) is an abstract software architecture, based on one or more domains, with no implementation aspects. Identified security threats can be controlled by adding security patterns to Cloud RA, which results in the Cloud Security RA (SRA). The VME is part of the Virtualization Layer of the cloud system, while the Container Environment is part of the cloud PaaS. See (Fernandez et al. 2016a) for the rest of the patterns in this figure, which are not relevant for this paper.

Author's address: Madiha H. Syed, email: msyed2014@fau.edu; Eduardo B. Fernandez, email: ed@cse.fau.edu; Dept. of Computer and Elect. Eng. and Computer Science Florida Atlantic University, Boca Raton, FL 33431, USA.

We describe here a Virtual Machine Environment, a system able to create and manage virtual machines according to user requests. Our audience are designers and system administrators of cloud systems or shared servers.

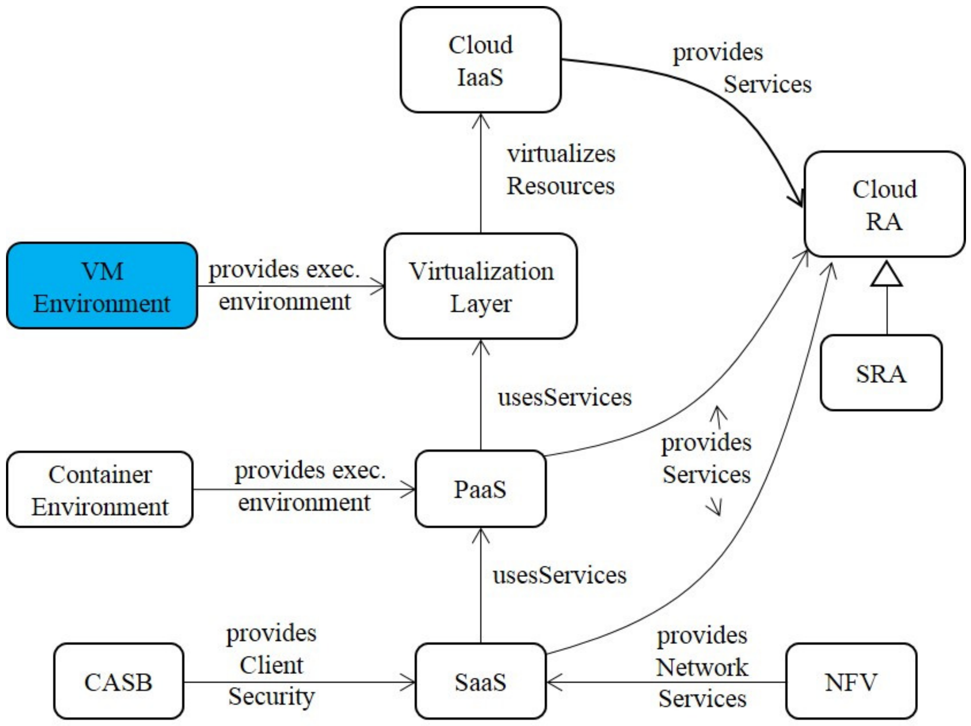

Fig. 1. Partial Cloud Ecosystem

2. VIRTUAL MACHINE ENVIRONMENT (VME) PATTERN

2.1 Intent

Provide units of execution (virtual machines), which have access to virtualized hardware (processor, memory, and networks). A VME creates and manages virtual machines according to user requests or system needs.

2.2 Example

A new company wants to develop application programs that need to run in Linux and Unix supported by different hardware platforms (instruction sets). Doing this development requires access to a variety of hardware, all of them running Linux and/or Unix. This is expensive and requires additional time and expertise to set up these processing environments. We just don't have enough money to do this work and we cannot develop the intended applications.

2.3 Context

Many companies and individuals need access to inexpensive processing power. Many companies develop software intended to run in multiple hardware platforms, and their software needs to be developed and tested in these platforms.

2.4 Problem

How do we provide inexpensive computing power to many users? Real processors can be purchased and used to host applications; however, this is expensive and most of the resources they provide will most likely go under-utilized. How do we develop software intended to run in multiple hardware instruction sets? We need an execution environment that can obtain these objectives conveniently and effectively.

The possible solution is constrained by the following forces:

– *Access to hardware features*: Some applications may need access to a specific set of hardware features to support their execution.

- *Flexibility*: we would like to run different types of operating systems or systems software in each execution instance.
- *Isolation*: There should be no way for a user in an execution environment to get access to the data or functions of another environment, either by error or intentionally. When an OS crashes or it is penetrated by a hacker, the effects of this situation should not propagate to other environments in the same hardware. Isolation is an important requisite for security and reliability.
- *Modularity:* execution instances should have a well-defined interface to the software that will execute on them; this can improve interoperability.
- *Usability*: Configuring an execution environment should be simple.
- *Portability*: We want execution instances to be portable across hardware processors; that is, they should be able to be moved or copied from one processor to another processor without special handling.
- *Extensibility*: It should be possible to dynamically provide additional services to the hosted applications like logging/auditing, filtering, persistence, and others.
- *Elasticity*: It should be possible to dynamically increase or reduce the resources needed by an execution instance. This is a useful feature for applications that need to handle variable loads.
- *Management:* It should be possible to manage execution environments to control the number of VMs, migration of VMs across processors for reliability and load balancing for optimal use of resources.
- *Monitoring:* It should be possible to monitor the state and activities of VMs. This could have value to assure SLAs.

2.5 Solution

Provide a platform, called Virtual Machine Environment (VME), to create and manage execution instances (processes), where application developers can think they are interacting with the real hardware. This virtualized processor, a Virtual Machine (VM), will have all the features of a hardware processor, except the same speed. VMs are useful to provide inexpensive computing power and variety of instruction set. VME provide a platform for creation, management, security, load balancing, monitoring, etc. of VMs. In this case, the software or even the hardware can be chosen so as to support the application and optimize or improve a non-functional requirement. A variety of software, e.g., databases, CAD/CAM, analytics packages, can be obtained from the Service Provider (SP) and loaded in the VM using a menu-like approach (Binaries/libraries).

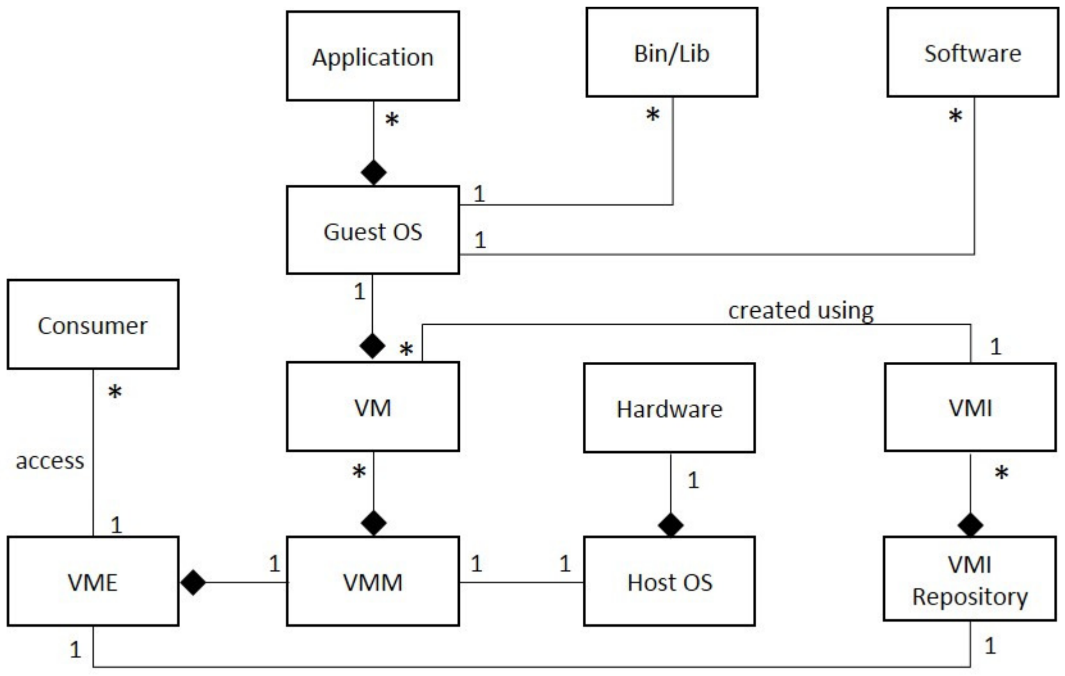

Fig. 2. UML Class diagram of a VM Environment

2.6 Structure

Figure 2 shows the class diagram of the **Virtual Machine Environment (VME)**. Consumers request the VME to create a **Virtual Machine (VM)** and provide a list of required resources. Each **VM** runs under the control of a **Virtual Machine Monitor (VMM)** (a.k.a Hypervisor), which provides a replica of all the features of some **Hardware** architecture. A **VMI Repository** is a collection of **VM Images (VMI)**, which are prepackaged software templates used by the VMM to instantiate VMs. We can run any type of **Guest Operating System** and **Binaries/Libraries** in each VM. The diagram shows a **Host OS** but this is not used in some implementations (see Section Implementation). The **VME** not only creates **VMs** but also performs management activities like monitoring, load balancing, and security.

2.7 Dynamics

Use cases for VMEs include: Creation and Deletion of VMs, Defining Access Control for VMs, Adding Resources to a VM, VM Migration, VM Replication, Load Balancing, Monitoring VMs, and others.

Use cases like Creation of a VM by a cloud system require use of the Infrastructure as a Service (IaaS) functions: a user needs an account and needs to specify a location, the VM is assigned to a node, etc. Several of these patterns are shown in (Fernandez 2013a); or conciseness we only show here the use case to create a VM by a user (Figure 4). The final objective for a user is to build a VM with a set of selected software units (Figure 3), typically an operating system, a DBMS, development tools, etc. The size of memory, location, and other features can be also selected, as well as access control rules.

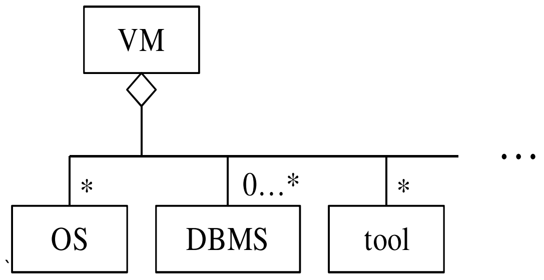

Fig. 3. Final objective of a VM creation

Figure 4 shows the sequence of steps performed when user creates a VM. For the creation of a VM, a user can create a VM image with the required set of resources or select an existing VMI from VMI repository. We assume here that the user selects an already existing VMI from the repository.

(1) A user requests VME for creating a virtual machine by providing a list of required resources.
(2) The VME selects a VMI from the repository based on user requirements.
(3) The VMI repository provides the requested image.
(4) The VME sends request to the VMM along with the VMI and list of resources to create a VM.
(5) The VMM instantiate the VMI to creates a VM with the requested resources
(6) The VMM assigns required hardware resources to the VM.
(7) The VM is assigned to the consumer and acknowledgement is sent by VME to them that the VM has been created.

As a post-condition of this use case, a VM is created and assigned to a user account and to the required hardware.

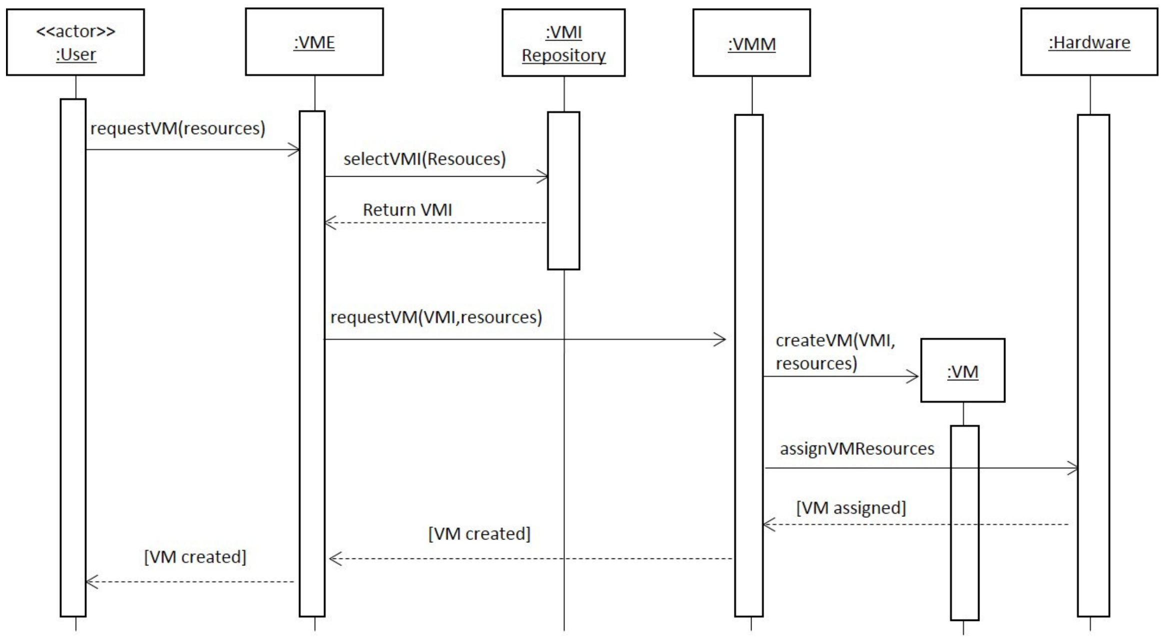

Fig. 4. Sequence diagram for the use case "Create a VM by a Use"r.

2.8 Implementation

VMs are created by a Virtual Machine Monitor (VMM). VMMs can run on real hardware (bare metal VMM) or on a host operating system (Type 1 and 2 hypervisors, respectively). In the latter case the I/O drivers of the guest operating systems run in the Host Operating System. This reduces memory needs and improves performance but it makes the virtual environment more complex which makes it more vulnerable to attacks. Figure 5 shows the stack of the VM execution environment with type 1 and 2 hypervisors.

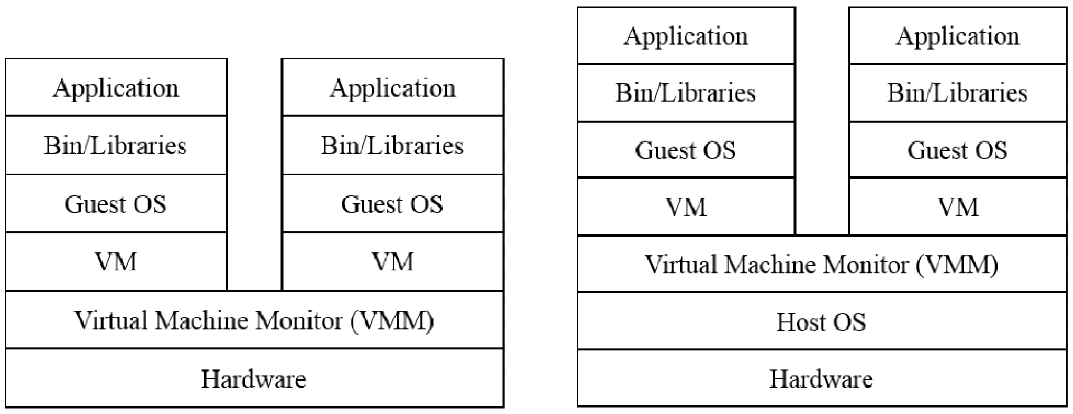

Figure 5. Stack of the VM execution environment. Bare metal VMM and Hosted VMM respectively

VMs can be created from Virtual Machine Images (VMIs) or in an ad hoc way by the user. VMIs are kept in a VMI Repository (Fernandez et al. 2013b). VMIs load VMs with different amounts of storage, specific software, or other features. Communication between VMs and users or external systems can be done using real or virtual networks. Virtual networks can use physical devices (switches, routers) or virtualized functions (NFVs) (Fernandez and Hamid 2015).

There are several types of virtualization which can have an influence on the speed and security of the corresponding VMs:

– *Full or Native Virtualization*—The guest operating system is presented with a full hardware instruction set and executes all instructions either directly or through the hypervisor. Some instructions do not trap and need extra binary code to be executed. This problem can be solved using Hardware-assisted Virtualization, where all privileged instructions trap to the hypervisor. This is the simplest approach as there is no need to modify the guest OS. So the Guest OS in figure 2 can be any standard OS (VMware 2008).

– *Paravirtualization*—This approach presents a software interface to VMs intended to reduce the portion of the guest's execution time spent performing operations which are substantially more difficult to run in a virtual environment compared to a non-virtualized environment (VMware 2008). It requires specially tailored guest operating systems running in the VMs. In class diagram for this type of VME, the Guest OS would provide customized instruction set.

– *Operating System Virtualization*—Allows the resources of a computer to be partitioned via kernel's support for multiple isolated user space instances, which are usually called containers and may look and feel like real machines to the end users. In case of OS virtualization, the virtualization environment would be simpler than VME. The components like Guest OS and VMM would no longer be required. Software containers are examples of OS virtualization (Fink 2014). It can be seen that containers operate with a lesser number of layers and therefore offer a lightweight execution environment as compared to VMs. However, since containers share the Host OS the resulting solution offers less flexibility (we cannot run together applications that need different operating systems). Of course, we can mix together in a cloud system VMs and containers.

– *Library Level Virtualization* - produces a different virtual environment working above the OS layer by implementing a specialized application binary/programming interface (ABI/API). Unikernels are examples of library level virtualization. Here the application code and runtime environment form a specialized kernel which is run directly on top of the hypervisor (Madhavapeddy and Scott 2013).

2.9 Example resolved

The company is now renting VMs from a service provider (SP) and only pays for the computation time they use. The SP has a variety of hardware environments and the company can switch from one to another in no time. The availability of different hardware architectures, the speed of change, and the flexibility of this approach results in shorter development times and lower costs. Now it has become feasible for this company to produce multi-platform applications.

2.10 Known uses

– IBM VM/370 (Creasy 1981). This was the first VMOS; it provided VMs for an IBM370 mainframe.
– VMware (Nieh and Leonard 2000). This is a commercial company that provides VMs for Intel x86 hardware. VMware ESX and ESXi are bare metal hypervisors
– Xen is a VMM for the Intel x86 developed as a project at the University of Cambridge, UK (Barham et al 2003).
– KVM (Kernel-based Virtual Machine) offers full virtualization for Linux on x86 systems (KVM 2017).
– QEMU (Quick Emulator) is an open-source VMM which runs on x86 systems (QEMU 2016). It can be run as a pure emulator or native VM.
– VirtualBox is a free and open-source VMM from Oracle for x86 hardware (VirtualBox 2016).
– Some smart phone operating systems use a few specialized virtual machines to separate the user’s private system from her work environment. These include the L4 Microvisor (Heiser and Leslie 2010) and RIM’s Black Berry 10 OS (Wikipedia 2016).
– OpenStack (Fernandez et al. 2016c) is one of the possible architectures for IaaS services that includes a VME. OpenStack supports KVM, VMware, Xen, Hyper-V, QEMU as hypervisor alternatives in addition to containers (LXC and Docker) (OpenStack 2017).

2.11 Consequences

The Virtual Machine Environment Pattern has the following advantages:

– *Access to hardware features*: Each operating system or other system software has access to a complete set of hardware features to support its execution. Not all users may require this type of access.

– *Flexibility:* we can run different types of operating systems or systems software in each VM.
– *Isolation:* The VMM intercepts and checks all system calls. The VMM is in effect a Reference Monitor (Fernandez 2013a, Gollmann 2006) and provides total mediation on the use of the hardware. This can provide a strong isolation between virtual machines (Rosenblum and Garfinkel 2005).
– *Modularity:* execution instances can have standards for their interface to the software that will execute on them; in fact, there is already an standard for this interface (OVF 2016)
– *Usability:* Configuring an execution environment requires only to select software packages from a menu.
– *Portability*: We can move a VM from one processor to another by just moving its process descriptor. We can also replicate VMs in this way to provide fault tolerance.
– *Extensibility*: It is possible to dynamically provide additional services.
– *Elasticity:* The SP has tools to dynamically increase or reduce the resources needed by a VM.
– *Management*: We can manage execution environments and perform load balancing when requests are received by VME.
– *Monitoring*: We can monitor the state of VMs and interactions of VMs and Guest OS (Bauman 2015).

The Virtual Machine Environment has the following liabilities:

– The rich functionality of the VME may not be needed for some applications; those applications should be developed in a Container Environment (Syed and Fernandez 2015).
– There is extra complexity and some type of usable dashboard is necessary to simplify the work of the system administrators, as it has been done in some cloud products, e.g. Fujitsu (Okuhara et al. 2011).

Related patterns

– Container (Syed and Fernandez 2015). A Software Container provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers. Containers have less execution overhead but are less flexible than VMs.
– Reference Monitor. As indicated, the VMM includes a concrete version of a Reference Monitor. A Reference Monitor is an abstract process that intercepts attempts to access resources and applies authorization rules (Fernandez 2013a, Gollmann 2006).
– Cloud Security Reference Architecture (Fernandez et al. 2016a). Indicates where the VME fits in the cloud architecture.
– NFV (Fernandez and Hamid 2015). Network Functions Virtualization (NFV) is an architecture for the construction of network services using software building blocks. The building blocks, Virtual Network Functions (VNFs), are typically created from cloud services using virtual machines or containers.
– Secure Virtual Machine Image Repository (Fernandez et al. 2013b): Avoid the poisoning of VM images during creation and the leaking sensitive information accidentally left in the VMI by enforcing access control to the repository.
– Cloud Policy Management Point (Fernandez et al. 2013b): Provide an administrative dashboard for security functions, including authentication, authorization, cryptography, logging, and control of VM images.
– IaaS patterns (Fernandez et al. 2016c) include several abstractions of services that include a VME.

3. CONCLUSION

Virtualization has long established itself as enabler of many technologies including the Cloud and IoT. VME enables the creation and management of virtual execution environments; it provides a way to virtualize cloud hardware and Infrastructure as a Service. This pattern is part of an on-going work on cloud ecosystems (Fernandez et al. 2016) which aims at developing a holistic and unified view of the Cloud to its users, designers, developers, and researchers, suitable to analyze aspects such as reliability and security. Many components of the system have already been modelled, some still have to be defined as patterns. As cloud computing evolves new components will be added and the cloud ecosystem model will also require updates and additions to reflect these changes. We are now working on architectural modeling of container clusters.

ACKNOWLEDGEMENTS

We thank Jason Yip for his comments that have improved this paper. Although he was not the official shepherd (our assigned shepherd never showed up), he provided many useful comments. The writers' group at PLoP'16 also provided useful comments.

REFERENCES

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *Proc. of the nineteenth ACM symp. on Operating systems principles* (SOSP '03). ACM, New York, NY, USA, 164-177. DOI=http://dx.doi.org/10.1145/945445.945462.

Erick Bauman, Gbadebo Ayoade, and Zhiqiang Lin. 2015. A Survey on Hypervisor-Based Monitoring: Approaches, Applications, and Evolutions. *J. ACM Comput. Surv.* 48, 1, Article 10 (August 2015), 33 pages. DOI: https://doi.org/10.1145/2775111

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. 1996. Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns. J. Wiley & Sons, Inc.

R. J. Creasy. 1981. The origin of the VM/370 time-sharing system. *IBM J. Res. Dev.* 25, 5 (September 1981), 483-490. DOI=http://dx.doi.org/10.1147/rd.255.0483

Eduardo B.Fernandez and T. Sorgente. 2005. A pattern language for secure operating system architectures. In *Procs. of the 5th Latin American Conference on Pattern Languages of Programs*. Campos do Jordao, Brazil. 16-19.

Eduardo B. Fernandez. 2013a. *Security patterns in practice: Designing Secure Architectures Using Software Patterns*. J. Wiley & Sons, Inc.

Eduardo B. Fernandez, Raul Monge, and Keiko Hashizume. 2013b. Two patterns for cloud computing: Secure Virtual Machine Image Repository and Cloud Policy Management Point. In *Proc. of 20th Conf. on Pattern Languages of Programs.* (PLoP 2013). The Hillside Group, Monticello, IL, Article 15, 11 pages.

Eduardo B. Fernandez and Brahim Hamid. 2015. A pattern for network functions virtualization. In *Proc. of the 20th European Conf. on Pattern Languages of Programs* (EuroPLoP '15). ACM, New York, NY, Article 47, 9 pages. DOI: http://dx.doi.org/10.1145/2855321.2855369

Eduardo. B. Fernandez, Raul Monge, and Keiko Hashizume. 2016a. Building a security reference architecture for cloud systems. *J. Requirements Engineering*. 21, 2 (June 2016), 255-249. Doi: 10.1007/s00766-014-0218-7

Eduardo B. Fernandez, N. Yoshioka, H. Washizaki and M. H. Syed. 2016b. Modeling and security in cloud ecosystems. *J. Future Internet* 2016, 8(2), 13; doi:10.3390/fi8020013 (Special Issue Security in Cloud Computing and Big Data)

Eduardo B. Fernandez, H. Washizaki and N. Yoshioka. 2016c. Patterns for Secure Cloud IaaS. In *Proc. of 5th Asian Conference on Pattern Languages of Programs* (AsianPLoP 2016).

J. Fink. 2014. Docker: a Software as a Service, Operating System-Level Virtualization Framework. *Code4Lib Journal*, 25 (July 2014). http://journal.code4lib.org/articles/9669

E. Gamma, R. Helm, R. Johnson, and J. Vlissides, 1994. *Design patterns –Elements of reusable object-oriented software* (1st. ed.). Addison-Wesley.

D. Gollmann. 2006. *Computer security* (2nd. ed.). Wiley.

Gernot Heiser and Ben Leslie. 2010. The OKL4 microvisor: convergence point of microkernels and hypervisors. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems* (APSys '10). ACM, New York, NY, USA, 19-24. DOI=http://dx.doi.org/10.1145/1851276.1851282

KVM. 2017. Kernel Virtual Machine. Retrieved February 10, 2017 from http://www.linux-kvm.org/page/Main_Page.

Anil Madhavapeddy and David J. Scott. 2013. Unikernels: Rise of the Virtual Library Operating System. *Queue* 11, 11 (December 2013), 30-44. DOI: http://dx.doi.org/10.1145/2557963.2566628

Jason Nieh and Ozgur Can Leonard. 2000. Examining VMware. *Dr. Dobb's Journal* 25, 8 (August 2000), 70 pages.

G. Nutt. 2003. *Operating systems* (3rd Ed.), Addison-Wesley.

M. Okuhara, T. Suzuki, T. Shiozaki, and M. Hattori, "Fujitsu approach to cloud-related information security", Fujitsu Sci. Tech. J., vol. 47, No 4, October 2011, 459-464.

OpenStack. 2017. OpenStack Compute Nodes. Retrieved February 10, 2017 from https://docs.openstack.org/ops-guide/arch-compute-nodes.html.

OVF. 2016. Open Virtualization Format. Retrieved July 15, 2016 from https://www.dmtf.org/standards/ovf

QEMU. 2016. Retrieved August 10, 2016 from https://en.wikibooks.org/wiki/QEMU

Mendel Rosenblum and Tal Garfinkel. 2005. Virtual Machine Monitors: Current Technology and Future Trends. *J. Computer* 38, 5 (May 2005), 39-47. DOI=http://dx.doi.org/10.1109/MC.2005.176

Madiha H. Syed and Eduardo B. Fernandez. 2015. The Software Container pattern. In *Proc. of 22nd Conference on Pattern Languages of Programs* (PLoP 2015). Pittsburgh, PA.

VirtualBox. 2016. VirtualBox – Oracle VM VirtualBox. Retrieved August 8, 2016 from https://www.virtualbox.org/wiki/VirtualBox

VMware. 2008. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Retrieved August 10, 2016 from http://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html

Wikipedia. 2016. Black Berry 10. Retrieved July 18, 2016 from https://en.wikipedia.org/wiki/BlackBerry_10