A Pattern for a Virtual Machine Environment

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

Keywords: virtual machine, architecture patterns, security, virtualization, security patterns, software container

Abstract

A Virtual Machine environment provides Virtual Machines which are isolated units of execution with access to virtualized hardware resources. VMs are created and managed by the VME upon user requests. 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 stacks in each VM. We present a pattern for a Virtual Machine Environment which describes benefits and liabilities of the VME. Alternate solutions for portable and faster execution environments have appeared, like software containers, but still VMs continue to be the most versatile and flexible among the virtualization solutions.

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 [Fer05 and Fer13]. Later, VMs were used to share expensive hardware, such as web servers, which hosted web sites isolated from other sites. Now they 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 [Sye15]; they trade off flexibility for speed. A Software *Container* provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers. Containers are lightweight, portable, extensible, reliable, and secure.

As shown in Figure 1, VMs and Containers are elements of cloud ecosystems [Fer16]. 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 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, this results in the Cloud Security RA (SRA). See [Fer16] for the rest of the patterns in this figure.

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

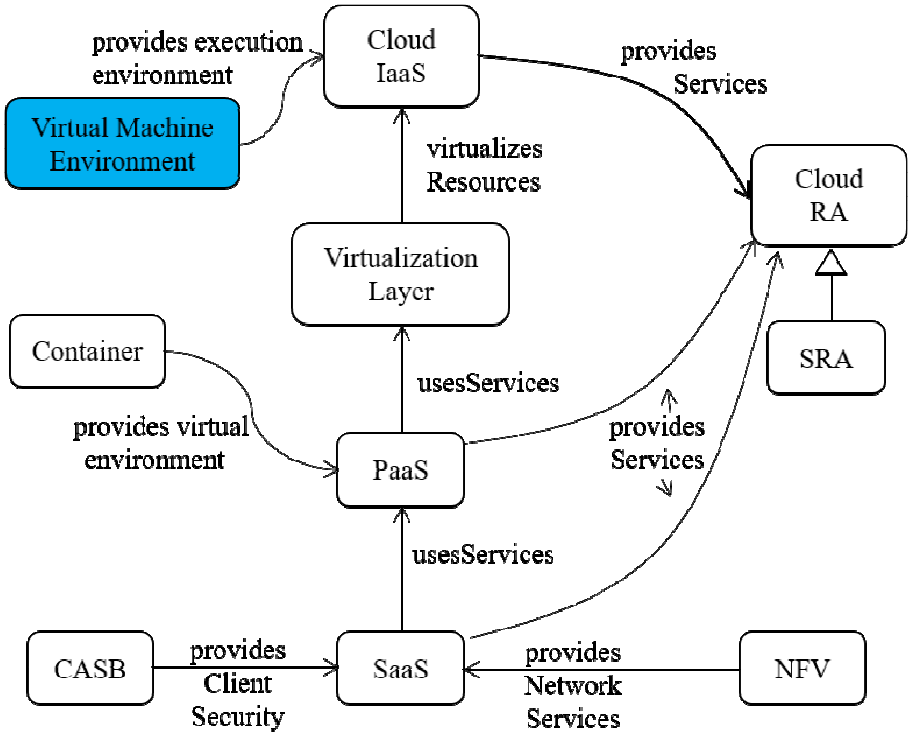

Figure 1. Partial Cloud Ecosystem

Virtual Machine Environment (VME)

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.

Example

A new company wants to develop application programs that need to run in Linux and Unix supported by different hardware platforms. 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.

Context

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

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.

The possible solution is constrained by the following **forces:**

Access to hardware features: Some applications may need access to a complete set of hardware features to support its 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 OSs in the same hardware.
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.

Solution

Create execution instances (processes) where application developers can think they are interacting with the real hardware, i.e., a virtualized processor with all the features of a hardware processor, this is a *Virtual Machine* (VM). 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).

Structure

Figure 2 shows the class diagram of the VM Environment. 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 show a **Host OS** but this is not used in some implementations (see Section Implementation).

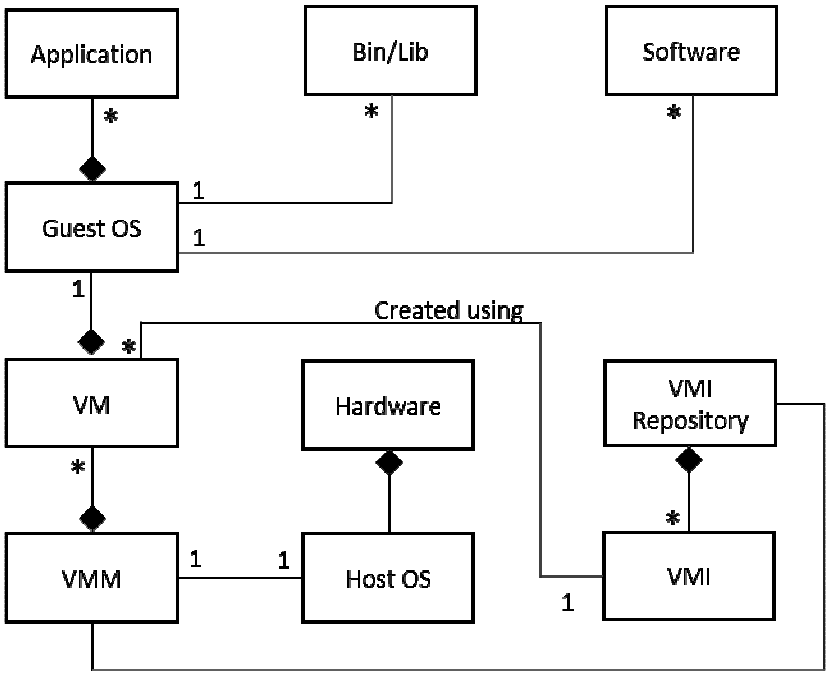

Figure 2. Class diagram of a VM Environment

Dynamics

Use cases for VMs include: Creation and Deletion of VMs, Selecting access control for VMs, Adding resources to a VM, VM Migration, VM Replication, and others.

Patterns like Creation of a VM by a cloud system require use of the IaaS functions: a user needs an account, the user needs to ask for a location, the VM is assigned to a node, etc. Several patterns of this type are shown in [Fer13] but we do not consider them as part of the VM Environment.

We show below a use case to create a VM by a user. 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. The sequence diagram of Figure 4 shows this use case.

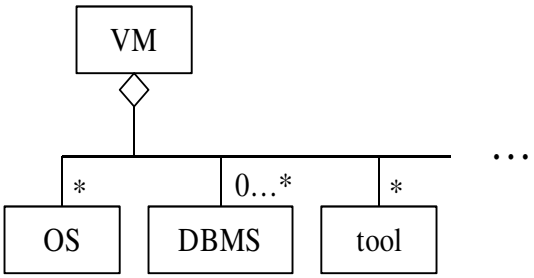

Figure 3. Final objective of a VM creation

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. For this use case, we assume that the user selects the already existing VMI from the repository.

1. A consumer selects a VMI from the repository based on required resources.
2. The VMI repository provides the requested image.
1. The consumer sends request to the VMM along with the VMI to create a VM.
3. The VMM instantiate the VMI to creates a VM with the requested resources
4. The VMM assigns required hardware resources to the VM.
2. The VM is assigned to the consumer and acknowledgement is sent to them that the VM has been created.

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

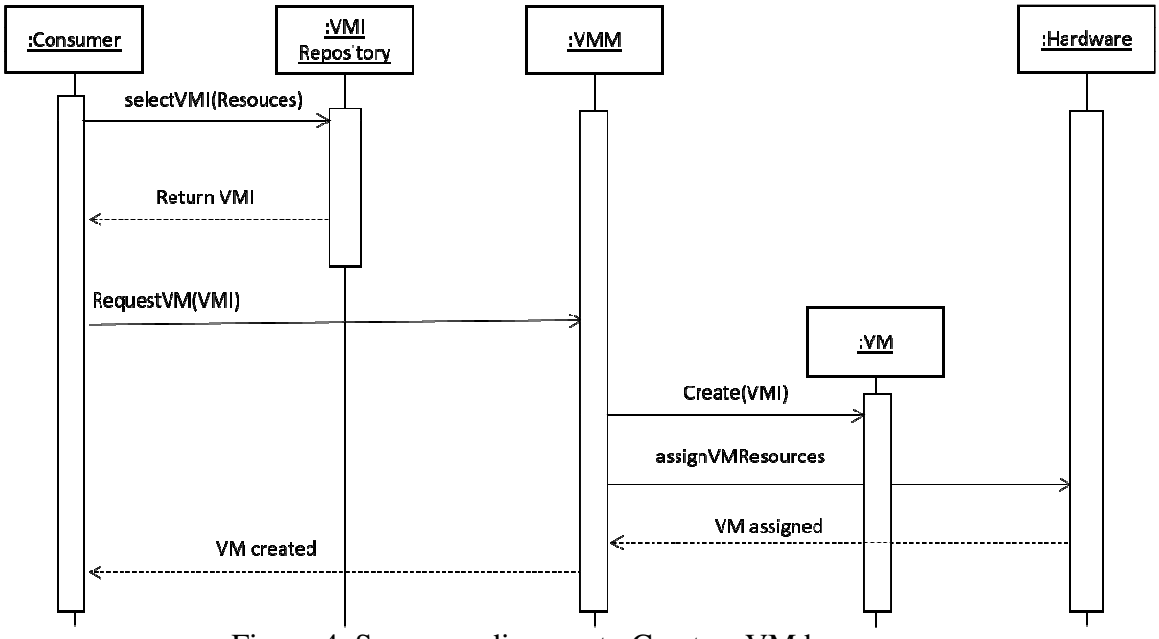

Figure 4. Sequence diagram to Create a VM by a user.

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 (Types 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.

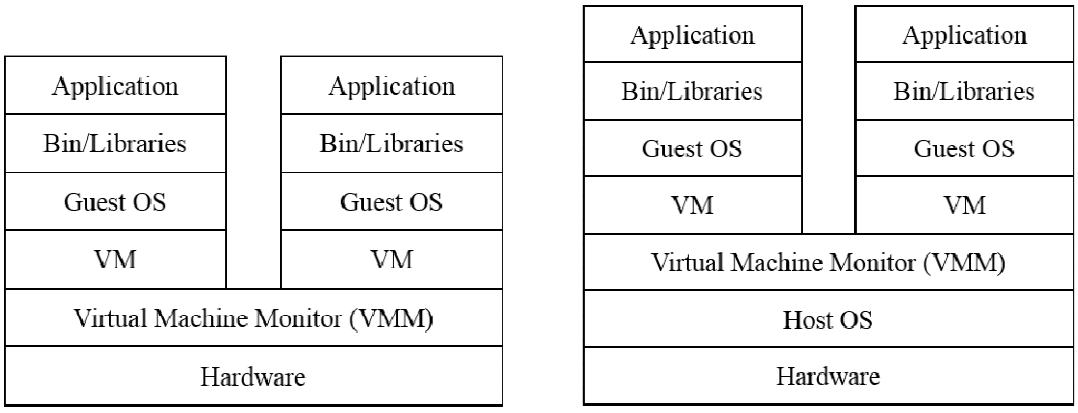

Figure 5. Stack of the VM execution environment. Bare metal VMM and Hosted VMM respectively

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 [Fer15]. 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 can be virtualized functions (NFVs) [Fer15b].

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.
Paravirtualization—Paravirtualization 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 (Wikipedia). It requires specially tailored guest operating systems running in the VMs.
Operating System Virtualization—allows the resources of a computer to be partitioned via the 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.
Library Level Virtualization - produces a different virtual environment working above the OS layer by implementing a specialized application binary/programming interface (ABI/API).

Figure 6 shows the stack for containers. 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.

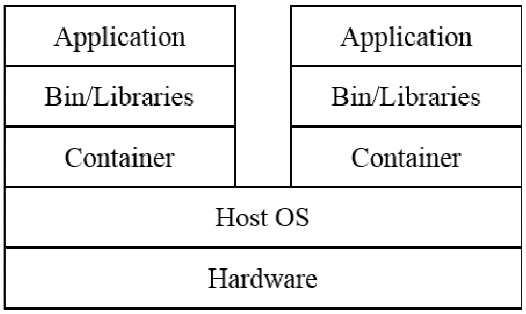

Figure 6. The Stack for a Software Container

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.

Known uses

- IBM VM/370 [Cre81]. This was the first VMOS, it provided VMs for an IBM370 mainframe.
- VMware [Nie00]. This company provides VMs for Intel x86 hardware.
- QEMU (Quick Emulator) is an open-source VMM which runs on x86 systems [Qem16]. It can be run as a pure emulator or native VM.
- Xen is a VME for the Intel x86 developed as a project at the University of Cambridge, UK [Bar00].
- VirtualBox is a free and open-source VME from Oracle for x86 hardware. [Vir16].
- 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 [Hei10] and RIM's Black Berry 10 OS [Bla16].

Consequences

The Virtual Machine 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 and provides total mediation on the use of the hardware. This can provide a strong isolation between virtual machines [Ros05].
- *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].
- *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.

The use of Virtual Machines has the following liabilities:

- There is a time overhead compared to running on a real processor.
- There is some time and complexity overhead compared to containers.
- There is a need to control the use of VMs for which we need an administration structure [Fer13b].

Related patterns

- Container [Sye15]. 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 [Fer13a].
- Cloud Security Reference Architecture [Fer15].
- NFV [Fer15b]. 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 [Fer13b]: 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 [Fer13b]: Provide an administrative dashboard for security functions, including authentication, authorization, cryptography, logging, and control of VM images.

Acknowledgements

We thank Jason Yip for his comments that have improved this paper. Although he was not the official shepherd, he provided many useful comments.

References

[Bar00] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization", Procs. of the ACM Symp. on Operating System Principles, SOSP'03.

[Bla16] Black Berry 10, https://en.wikipedia.org/wiki/BlackBerry_10

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns, Volume 1. Wiley, 1996.

[Fer05] E.B.Fernandez and T. Sorgente, "A pattern language for secure operating system architectures", Procs. of the 5th Latin American Conference on Pattern Languages of Programs, Campos do Jordao, Brazil, August 16-19, 2005.

[Fer13a] E.B.Fernandez, "Security patterns in practice: Building secure architectures using software patterns", Wiley Series on Software Design Patterns, 2013.

[Fer13b] E.B. Fernandez, Raul Monge, and Keiko Hashizume, "Two patterns for cloud computing: Secure Virtual Machine Image Repository and Cloud Policy Management Point", 20th Conf. on Pattern Languages of Programs (PLoP 2013)

[Fer15a] E.B.Fernandez, Raul Monge, and Keiko Hashizume, "Building a security reference architecture for cloud systems", Requirements Engineering. Doi: 10.1007/s00766-014-0218-7, 2015

[Fer15b] E. Fernandez and Brahim Hamid, "A pattern for Network Functions Virtualization", 21st European Conf. on Pattern Languages of Programs (EuroPLoP 2015)

[Fer16] E.B. Fernandez, N. Yoshioka, H. Washizaki and M.H.Syed, "Modeling and security in cloud ecosystems", *Future Internet* 2016, *8*(2), 13; doi:10.3390/fi8020013
(Special Issue Security in Cloud Computing and Big Data)

[Gam94] E. Gamma, R. Helm,R. Johnson, and J. Vlissides, Design patterns –Elements of reusable object-oriented software, Addison-Wesley 1994.

[Gol06] D. Gollmann, Computer security, 2nd Ed., Wiley, 2006.

[Hei10] Gernot Heiser and Ben Leslie. "The OKL4 Microvisor: Convergence point of microkernels and hypervisors". Proceedings of the 1st Asia-Pacific Workshop on Systems (APSys), pages 19–24, New Delhi, India, August 2010.

[Nut03] G. Nutt, Operating systems (3rd Ed.), Addison-Wesley, 2003.

[OVF] https://en.wikipedia.org/wiki/Open_Virtualization_Format

[Qem16] QEMU. 2016. https://en.wikibooks.org/wiki/QEMU. Accessed: 08/10/2016.

[Ros05] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends", Computer, IEEE May 2005, 39-47.

[Sch06] M. Schumacher, E.B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns: Integrating security and systems engineering, J. Wiley & Sons, 2006.

[Sil08] A. Silberschatz, P. Galvin, G. Gagne, Operating System Concepts (8th Ed.), John Wiley & Sons, 2008.

[Sye15] Madiha H. Syed and Eduardo B. Fernandez, "The Software Container pattern", 22nd Conference on Pattern Languages of Programs 2015, Pittsburgh, PA, October 24-26, 2015

[Tan08] A. S.Tanenbaum, Modern Operating Systems (3rd Ed.), Prentice Hall, 2008.

[Vir16] VirtualBox. 2016. https://www.virtualbox.org/wiki/VirtualBox. Accessed: 08/10/2016

[Wikipedia} https://en.wikipedia.org/wiki/Paravirtualization

,