MINES Saint-Étienne | Institut Mines-Télécom

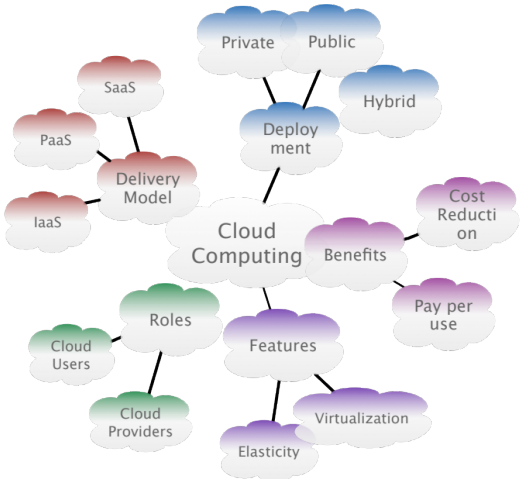
Virtualization – VMs and Containers

Luis Gustavo Nardin
gnardin@emse.fr

Cloud and Edge Infrastructures

November 9, 2025

Cloud Computing Overview

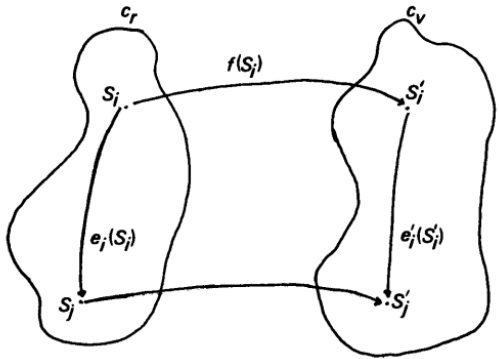


Outline

Virtualization

Virtualization

- Create a virtual version of something
  - Hardware, Network, Storage, Operating System, Application
- The construction of an **isomorphism** between a **guest system** and a **host**



(Popek & Goldberg, 1974)

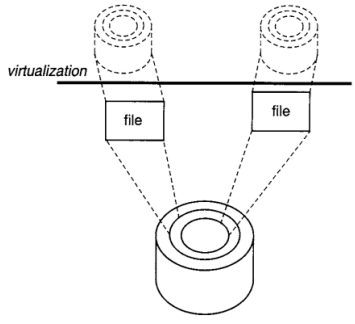
Virtualization

- **Virtual Machine Monitor (VMM)** is the software that provides the abstraction of a virtual machine
- VMM's essential characteristics are
  - **Equivalence / Fidelity**
    - ✓ Provide an environment for programs which is essentially identical with the original physical machine
  - **Resource Control / Safety**
    - ✓ Complete control of system resources
  - **Efficiency / Performance**
    - ✓ Programs running in this environment show at worst only minor decrease in speed

(Popek & Goldberg, 1974)

Example: Virtual Disk

- Partition a single hard disk to multiple virtual disks
- Virtual disk has virtual tracks & sectors
- Implement virtual disk by file
- Map between virtual disk and real disk contents
- Virtual disk write/read mapped to file write/read in host system



(Smith & Nair, 2005)

Interface Abstraction Levels

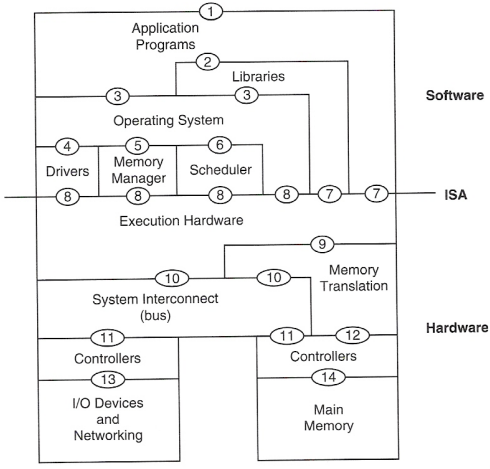


Figure 1.4 Computer System Architectures. *Implementation layers communicate vertically via the shown interfaces. This view of architecture is styled after one given by Glenford Myers (1982).*

(Smith & Nair, 2005)

Virtualization at Various Abstraction Levels

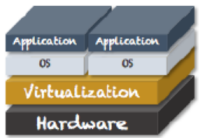
Level of Abstraction	Functional Description	Example Packages	Relative Merits, Appl. Flexibility/Isolation, and Implementation Complexity
Instruction Set Architecture	Emulation of a guest ISA by the host ISA	Dynamo, Bird, Bochs, Crusoe	Very low performance, high app flexibility, and median complexity and isolation
Hardware-Level Virtualization	Virtualization on top of bare metal hardware	XEN, VMWare, Virtual PC	High performance and complexity, median app flexibility, and good app isolation
Operation System Level	Isolated containers as OS instances	Docker Engine, Jail, FVM	Highest performance, low app flexibility and isolation, and average complexity
Run-Time Library Level	Creating VM via run-time library through API hooks	Wine, cCUDA, WABI, LxRun	Average performance, low app flexibility and isolation, and low complexity
User Application Level	Deploy HLL VMs at user application level	JVM, .NET CLR, Panot	Low performance and app flexibility, very high complexity and app isolation

(Hwang, 2017)

Virtual Machines

Virtual Machines

- **Virtual Machine (VM)**: an **efficient**, **isolated duplicate** of the real machine
- VMs run on top of a physical machine using a **Hypervisor**
- **Hypervisor**: a tool that controls and distributes the computing resources to each VM
- Roles of the hypervisor
  - Provide control of the processor and the resources of the host machine
  - Allocate to each virtual machine the resources requested
  - Make sure that VMs do not interfere with each other



Virtual Machine Monitor
(VMM, Hypervisor)

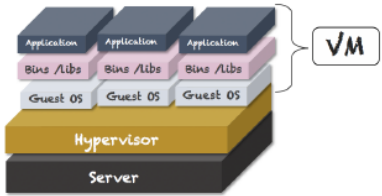
VM Advantages

- **Cost-Effectiveness – Less Hardware**
  - Multiple virtual machines / operating systems / services on single physical machine (server consolidation)
  - Various forms of computation as a service
- **Isolation**
  - Good for security
  - Great for reliability and recovery: If VM crashes it can be rebooted, does not affect other services (fault containment)
  - VM migration
- **Development Tool**
  - Work on multiple OS in parallel
  - Develop and debug OS in user mode
  - Origins of VMware as a tool for developers

(Hennessy & Patterson, 2019)

Bare Metal Hypervisor (Type I)

- A bare metal hypervisor is directly executed on the hardware
  - Interface directly with the underlying hardware
  - Do not need a host OS to run on



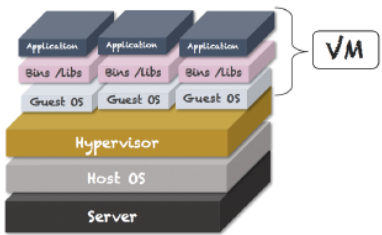
\+ Better performance, scalability, and stability

\- Hardware compatibility limited

Based on slides of Charlotte Laclau – Télécom Saint-Étienne

Host Metal Hypervisor (Type II)

- A hosted virtualization hypervisor runs on the OS of the hosting computer
  - A computer running OSX can have a VM running Windows
  - The VM does not have direct access to hardware, so it has to go through the host operating system

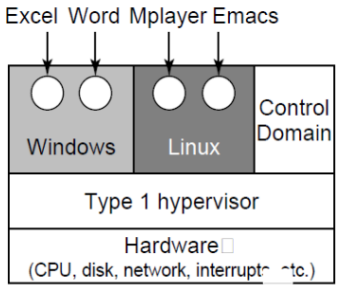


\+ More hardware compatibility

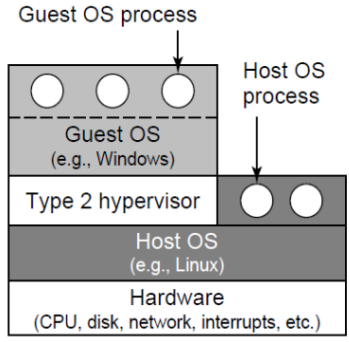
\- Lower performance

Based on slides of Charlotte Laclau – Télécom Saint-Étienne

Hypervisors Type I and Type II



(a)



(b)

(Hwang, 2017)

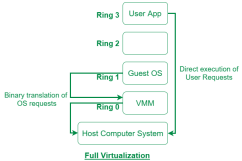
Full Virtualization vs. Paravirtualization

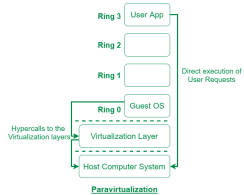
Full Virtualization

- **Do not modify the guest OS**, and critical **instructions are emulated** by software through the use of **binary translation**
- Binary translation **slows down the performance** considerably

Paravirtualization

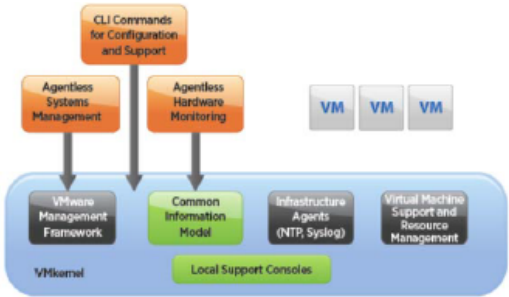
- **Modify the guest OS**, and non-virtualizable instructions are replaced by *hypercalls* that communicate directly with the hypervisor or VMM
- **Reduce the overhead**, but the cost of maintaining para-virtualized OS is high





Examples Bare Metal Hypervisors

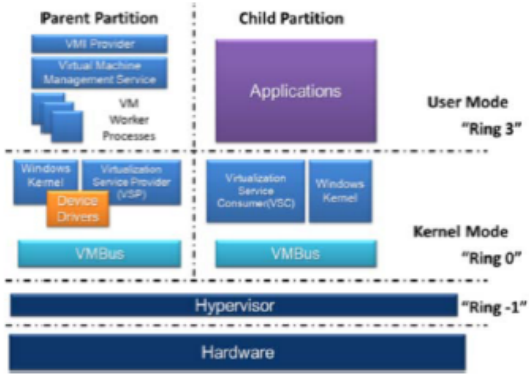
- **VMware ESXi**
  - Mature and stable tool
  - Small disk footprint size
  - Memory ballooning
  - Include its own kernel
  - Free edition with limited features



(Fayyad-Kazan et al., 2013)

Examples Bare Metal Hypervisors

- **Microsoft Hyper-V**
  - Good for Small-Medium Business
  - Simple live migrations
  - Good for running Windows
  - Free edition with limited features

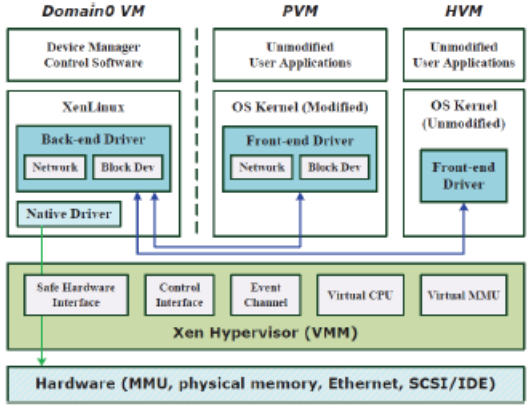


NOTE: Microsoft is ending mainstream support of Hyper-V Server 2019 on January 9, 2024 and extended support will end on January 9, 2029. Hyper-V Server 2019 will be the last version of this product and Microsoft is encouraging customers to transition to Azure Stack HCI.

(Fayyad-Kazan et al., 2013)

Examples Bare Metal Hypervisors

- **Xen Project**
  - Mature and stable tool
  - Open source
  - Micro-kernel hypervisor (Modular)
  - OS directly accesses physical devices
  - Domain 0 implements access policies



(Fayyad-Kazan et al., 2013)

Examples Host Metal Hypervisors

- **VMware Workstation Pro/Player**
  - Support multiple different OS
  - Good for labs and demonstrations
  - VMware Workstation Player free

- **Oracle VM VirtualBox**
  - Mature and stable tool
  - Open source
  - Suitable for SME
  - Support multiple OS

Examples Host Metal Hypervisors

- **KVM**
  - Open source
  - Integrated to Linux Kernel
  - Support wide variety of hardware
  - Live migration

- **Parallels**
  - Mature and stable tool
  - Run on MacOS (Intel and ARM processors)
  - Support multiple OS
  - Good for running Windows on MacOS

How to choose a hypervisor?

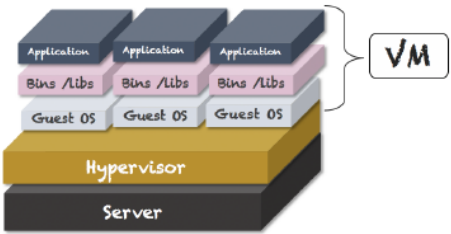
- Understand your needs, i.e., flexibility, scalability, reliable support, etc.
- Understand the features, i.e., live migration, storage migration, dynamic memory, etc.
- Investigate the ecosystem
  - Possible to evaluate every virtualization hypervisor for free
- Compare costs

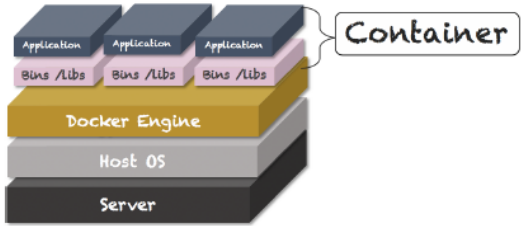
Based on slides of Charlotte Laclau – Télécom Saint-Étienne

Containers

Containers

- **Containers** provide an operating system level virtualization
- An abstraction layer between traditional OS and user applications
- **Main difference**: Containers share the host OS kernel with other containers





Based on slides of Charlotte Laclau – Télécom Saint-Étienne

Containers

Advantages

- Minimal startup/shutdown cost
- Small footprint
- High scalability

Disadvantages

- All containers must use the same OS
- Poor application isolation

Similarities with VMs

- They have private space for processing
- They can execute commands as superuser
- They have a private network interface and IP address
- They can mount filesystems

Hypervisors vs. Containers

Hypervisors

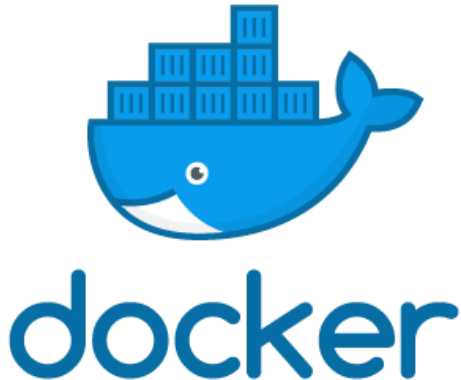
- Allow an OS to run independently from the underlying hardware through the use of virtual machines
- Share virtual computing, storage and memory resources
- Can run multiple operating systems

Containers

- Allow applications to run in separate divisions of the Host OS
- Can run on different OS, all they need is a container engine to run
- Extremely portable since in a container, an application has everything it needs to run

What is Docker?

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system–level virtualization on Linux



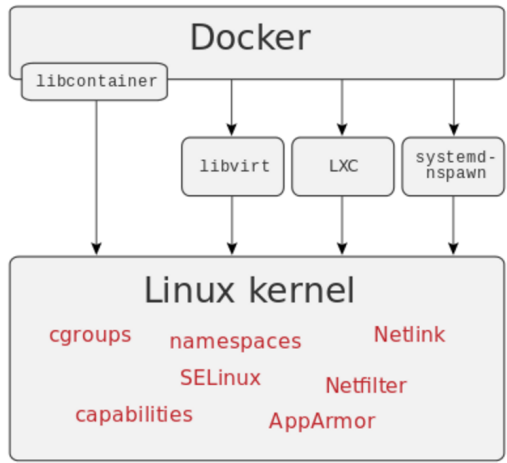
Source: https://en.wikipedia.org/wiki/Docker_(software)

Why Docker?

- **Ease of use**
  - can quickly build and test portable applications
  - allow anyone to package an application on their laptop
  - build once, run anywhere
- **Speed**: lightweight and fast
  - use fewer resources
  - no need to boot up a full virtual OS every time
- **Docker Hub**: an increasingly rich ecosystem
  - an App store for Docker images
  - public images created by the community
- **Modularity and Scalability**
  - easy to link containers together to create an app
  - easy to scale or update components independently in the future

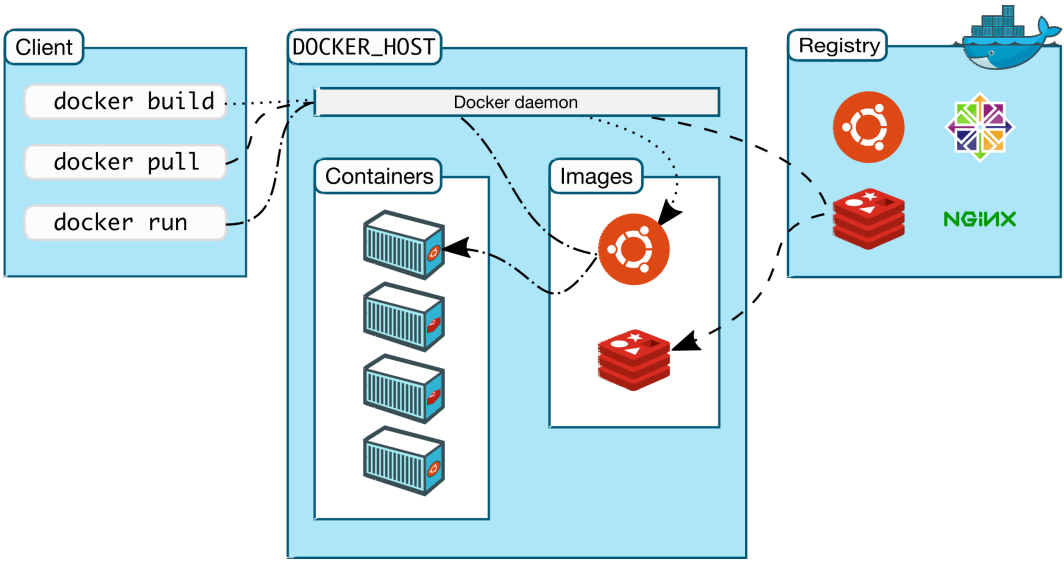
Docker Technology

- **libcontainer**: A native Go implementation for creating containers with namespaces, cgroups, capabilities, and filesystem access controls
- **libvirt**: Manage platform virtualization
- **LXC** (LinuX Containers): Multiple isolated Linux systems (containers) on a single host
- **systemd-nspawn**: Fully virtualizes the file system hierarchy



Source: https://en.wikipedia.org/wiki/Docker_(software)

Docker Architecture Overview



Docker Architecture Overview

- **Docker engine**: Layer on which Docker runs
  - **Docker Daemon** runs in the host computer
  - **Docker Client** communicates with the Docker Daemon to execute commands
- **Dockerfile**: Instructions to build a Docker image
- **Docker Image**: Read-only templates built from a set of instructions written in your Dockerfile

Kubernetes Overview

- What if I want to run multiple containers across multiple machines?
  - Need to start the right containers at the right time
  - Figure out how they can talk to each other
  - Handle storage considerations
  - Deal with failed containers or hardware

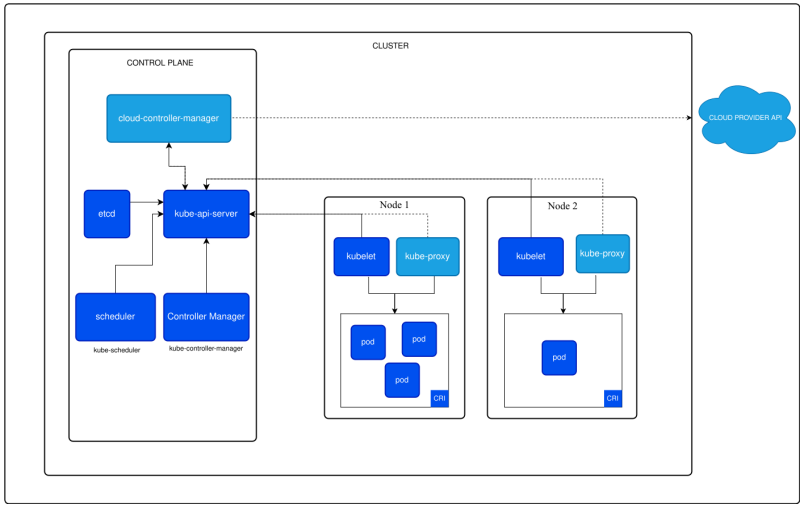
Based on slides of Charlotte Laclau – Télécom Saint-Étienne

Kubernetes Overview

- Open source container orchestration platform
  - Run containers across many different machines
  - Scale up or down by adding or removing containers when demand changes
  - Keep storage consistent with multiple instances of an application
  - Distribute load among containers
  - Launch new containers on different machines if something fails

Based on slides of Charlotte Laclau – Télécom Saint-Étienne

Kubernetes Architecture



Source: https://kubernetes.io/

Kubernetes Control Panel Concepts

kube-apiserver The API server that exposes the Kubernetes API.

etcd Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

kube-scheduler Watch for newly created Pods with no assigned node, and selects a node for them to run on.

kube-controller-manager Component that runs controller processes

Node Responsible for noticing and responding when nodes go down.

Job Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

EndpointSlice Populates EndpointSlice objects (to provide a link between Services and Pods).

ServiceAccount Create default ServiceAccounts for new namespaces.

cloud-controller-manager Embed cloud-specific control logic.

Kubernetes Control Panel Concepts

kubelet An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

kube-proxy kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

Container runtime A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

References

- Fayyad-Kazan, H., Perneel, L. & Timmerman, M. (2013). Benchmarking the Performance of Microsoft Hyper-V server, VMware ESXi and Xen Hypervisors. Journal of Emerging Trends in Computing and Information Systems, 4(12), pp. 922-933.
- Hennessy, J. & Patterson, D. (2019). Computer Architecture: A Quantitative Approach. (6th ed.). Amsterdam: Morgan Kaufmann.
- Hwang, K. (2017). Cloud Computing for Machine Learning and Cognitive Applications. Cambridge, MA: The MIT Press.
- Popek, G. & Goldberg, R. P. (1974). Formal Requirements for Virtualizable Third Generation Architectures. Communications of the ACM. 17(7), pp. 412-421. DOI: 10.1145/361011.361073.
- Smith, J. E. & Nair, R. (2005). Virtual Machines. San Francisco, CA: Morgan Kaufmann.