Principals of Mobile Cloud Computing

Part 1: Server Virtualization

Cloud-native Software-defined Mobile Networks
Department of Electrical Engineering
Sharif University of Technology
Azad Ravanshid

Section 1

FUNDAMENTALS OF SERVER VIRTUALIZATION

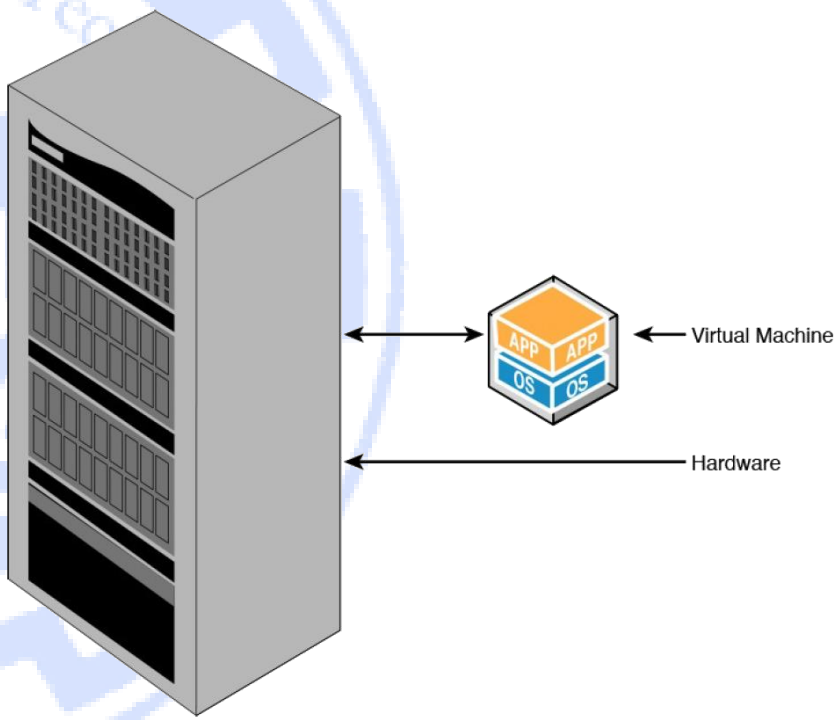
Virtualization Areas

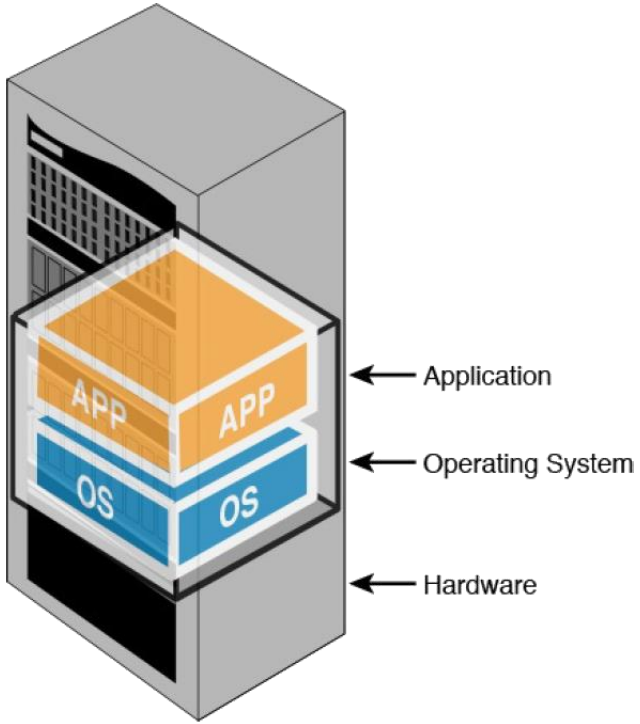
- Server Virtualization
- Network Virtualization
- Network Function Virtualization

Server Virtualization

- **Server Virtualization Concepts**
  - Used to run multiple operating systems (OSs) or applications on top of a single physical infrastructure by providing each of them an abstract view of the hardware.
  - Enabling these applications or OSs to run in *isolation* while sharing the same hardware resources
  - Each application is made to believe that it owns the hardware resources but it's not necessarily aware that this hardware is a subset abstracted from a bigger hardware pool

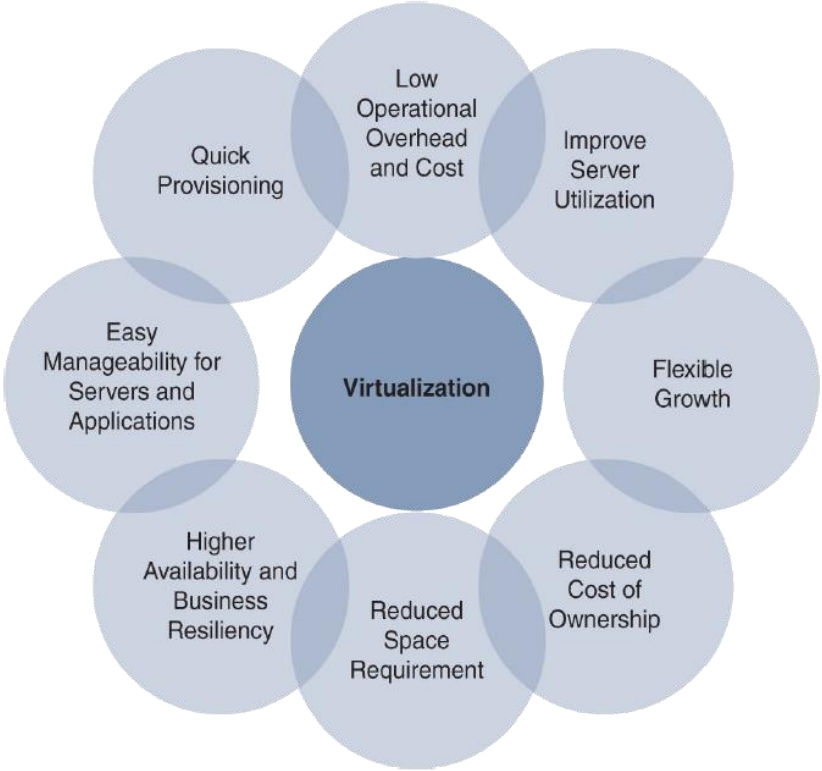
Sever Virtualization

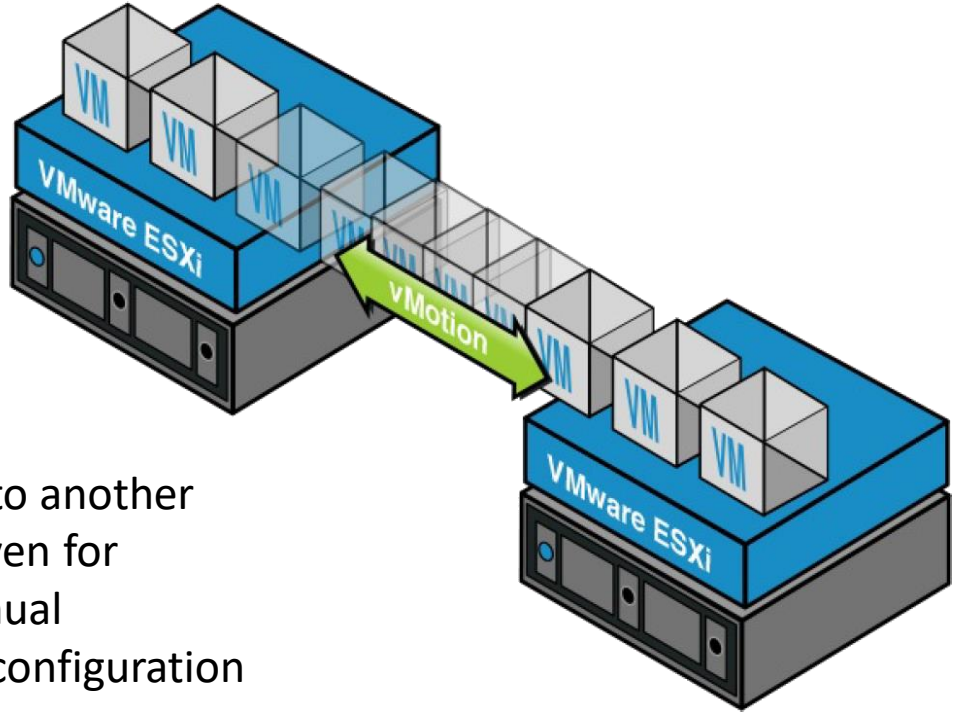




Virtualization Benefits

- Drastically reducing the number of servers doing the same job and simultaneously hosting multiple applications resulting in
  - Savings of Power, Space, and Operational Costs
  - Faster Application Spin-up and Provisioning
  - Higher Availability and Business Resiliency

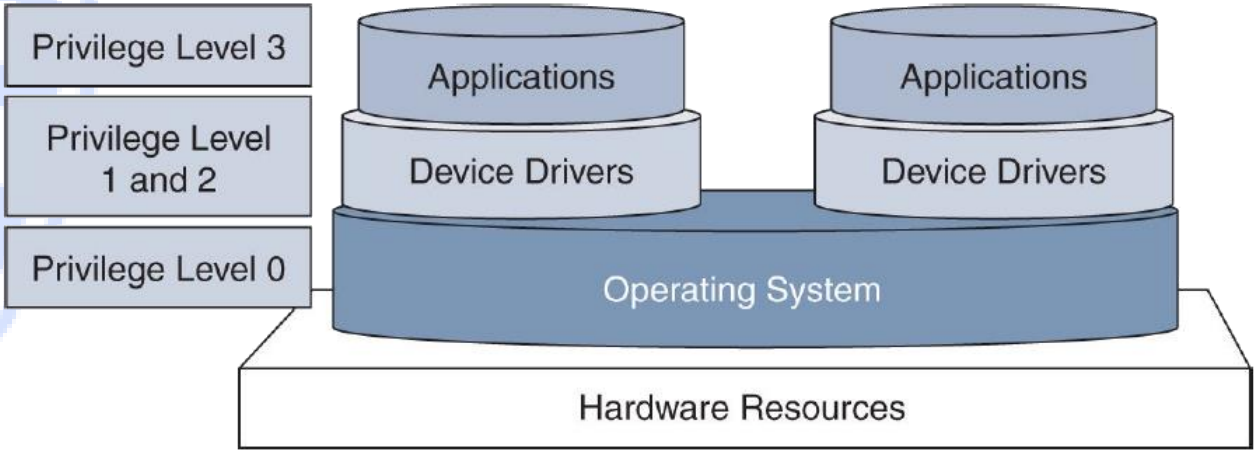




Possibility to seamlessly fail over to another VM instance on another server even for running services without any manual intervention or horribly complex configuration changes.

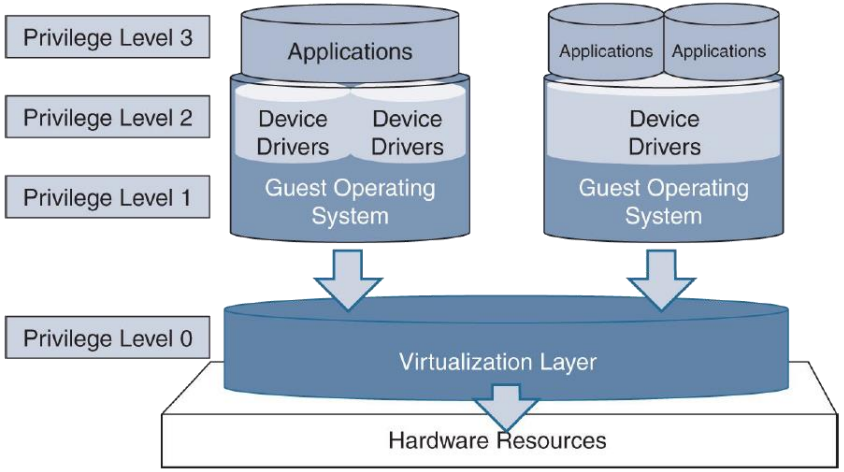
Virtualization Techniques

- **High-Level View of x86 Architecture**
  - The operating system needs to be in privilege level 0, so that it can interact directly with the hardware.



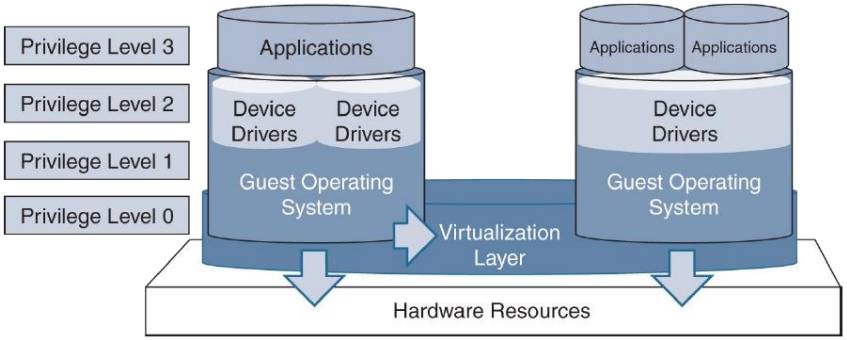
Virtualization Techniques

- **Full Virtualization**
  - The operating system (OS) is moved to the higher layer and the virtualization layer now occupies Layer 0
  - OS instructions to the hardware are now translated by the virtualization layer and in-turn sent to the hardware
  - cost associated and inefficient for esp. delay-critical applications due to two-layerd translation of the data from the OS to the virtualization layer and then to the hardware



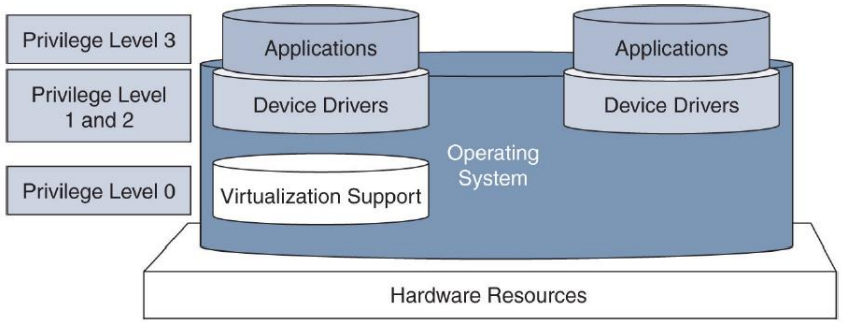
Virtualization Techniques

- **Para-virtualization**
  - The virtualization layer works along the guest operating systems at the same privilege level (layer 0) to execute time-critical functions
  - All the guest OSs running on the hardware need to be aware of each other to be able to share the resource.
  - Due to this requirement of the interaction with other guest OS as well as the virtualization layer, there is development effort needed on the guest operating system side to work in this environment.



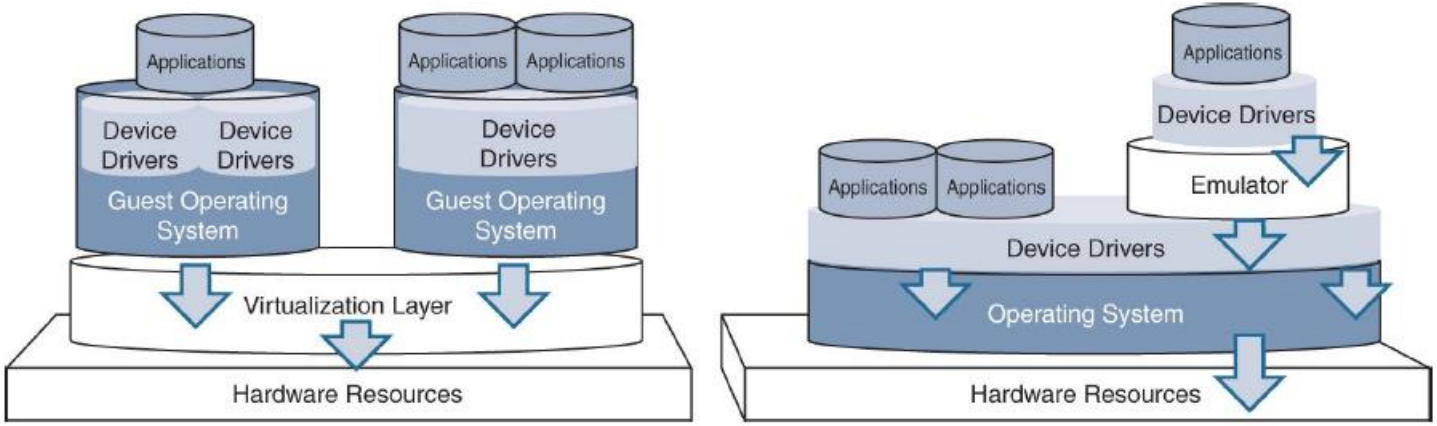
Virtualization Techniques

- **OS-level Virtualization**
  - Running applications directly on the single common
  - The OS is modified to give these applications their own working space and resources.
  - It ensures separation between the application's resource usage, making it look like they are still running on independent standalone servers.
  - Examples of this virtualization technique are Linux containers and Docker, which are discussed in depth later



Virtualization Techniques

- **Virtualization vs. Emulation**



How virtualization works

Virtual Machine (VM)

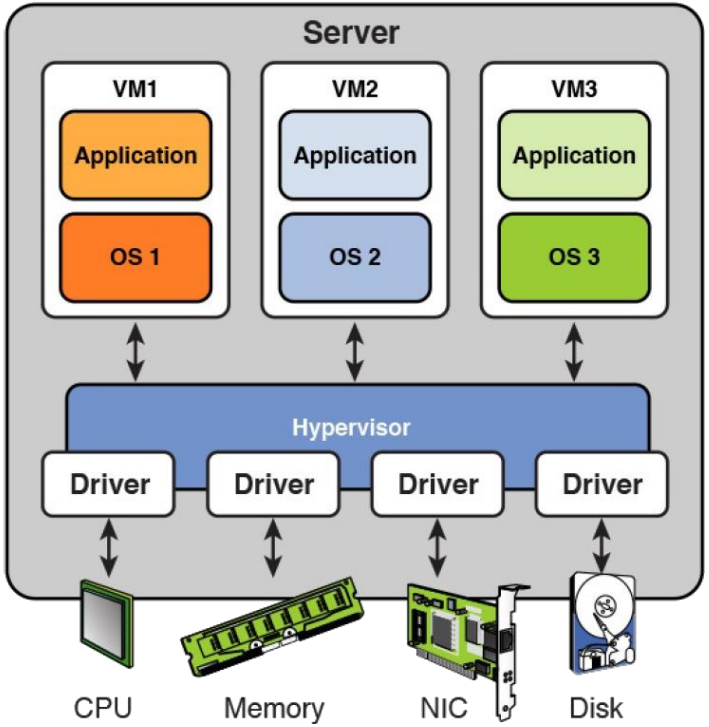
An isolated virtual hardware environment for an operating system or application to run upon

- **VM Components**
  - Host operating system
  - Hypervisor
  - Guest Operating System

How virtualization works

Hypervisor

piece of software that sits between the OS and the server ’s hardware resources to manage the connections between the drivers and the server ’s resources

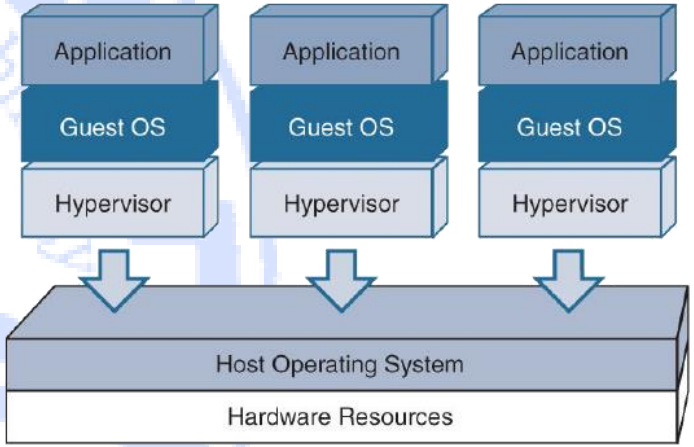


How virtualization works

- **Hypervisor Type I**
  - Direct interaction with the hardware; known as ***bare-metal* hypervisor**
  - Running directly on the hardware without the need for an underlying OS
  - VMWare ESXi, Microsoft Hyper-V, XEN (Citrix, open source) and KVM (?)

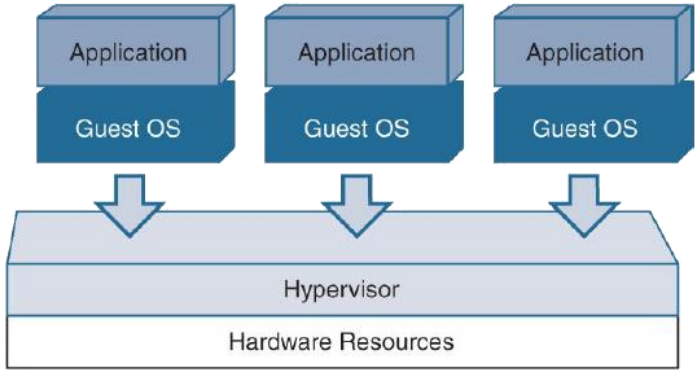
- **Hypervisor Type II**
  - Running as an application on the host OS, along with other applications and processes, and has specific hardware resources allocated to it.
  - Host OS provides a platform for the hypervisor to run. Additionally, it provides the interface to communicate with the devices, memory, disks, and other peripherals; known as ***hosted-hypervisor***
  - Host OS should be a quite thin and lightweight layer, fulfilling its basic function and conserving most of the resources for the guests (virtual machines) to utilize
  - VMWare Workstation and Fusion, Oracle Virtual Box and Qemu (open source)

Hypervisor Comparison



Type-2 Hypervisor

Qemu,
VMWare Workstation
Oracle VirtualBox
....



Type-1 Hypervisor

VMWare ESXi
Microsoft Hyper-V
Xen
....

Hypervisor Comparison

Type I

Due to direct interaction with the hardware:

- ✓ Higher performance w.r.s.t reduced communication overhead and *OS-independency*
- ✓ Self-contained and more secure
- × Increased complexity of the hypervisor code
- × Higher time to develop

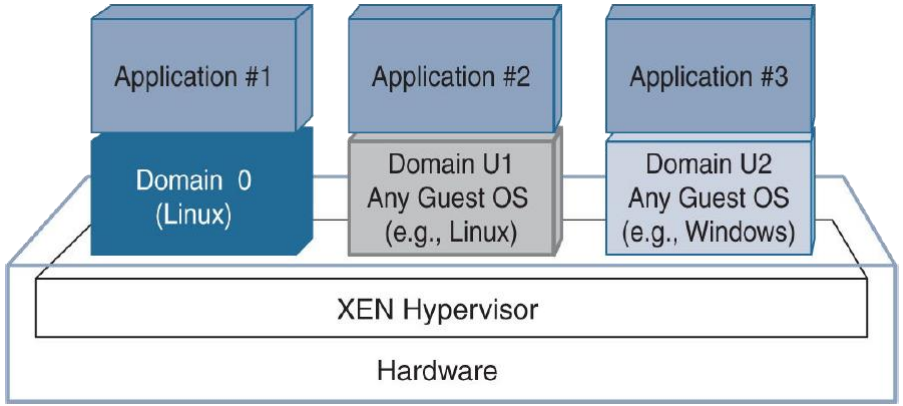
Type II

Due to relying on underplaying OS:

- ✓ Easier and quicker development
- ✓ More flexibility regarding *hardware-independency*
- ✓ Supporting wider range of hardware
- × Lower performance due to OS-dependency

Open Source Virt. Architecture

- **XEN**
  - An open source virtualization software and offers a type-1 hypervisor.
  - Dom0 has direct access to the hardware and manages the device drivers on the hardware, in addition to running an application that is being virtualized.
  - Based on *para-virtualization* technique, where the guest virtual machines request the hardware resource to be sent to Dom0 on an as-needed basis
  - With this additional requirement for the guest OS to interact with Dom0, special device drivers are needed on the guest OS.
  - The guest operating systems are made aware of the fact that they are not running on their own dedicated hardware. This requires some modification of the guest OS



Open Source Virt. Architecture

- **KVM (Kernel-based Virtual Machine)**
  - Type I hypervisor !!?
  - KVM enables mapping of physical CPU to Virtual CPU by translating the instructions meant for the vCPU and convert it into instructions for the physical CPU. This mapping provides the hardware acceleration for Virtual Machine and boosts its performance.
  - Live Migration and emulation not supported

Open Source Virt. Architecture

- **QEMU (Quick Emulator)**
  - A generic and open source machine emulator and virtualizer
  - QEMU by itself is a Type-2 hypervisor intercepts the instructions meant for Virtual CPU and uses the host OS to get those instructions executed on the physical CPU.
  - When used as a machine emulator, QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC).
  - QEMU supports virtualization when using the KVM.
  - Actually KVM arbitrates access to CPU and memory and helps QEMU for **hardware acceleration**, the combination becomes a Type-1 hypervisor

KVM vs. QEMU

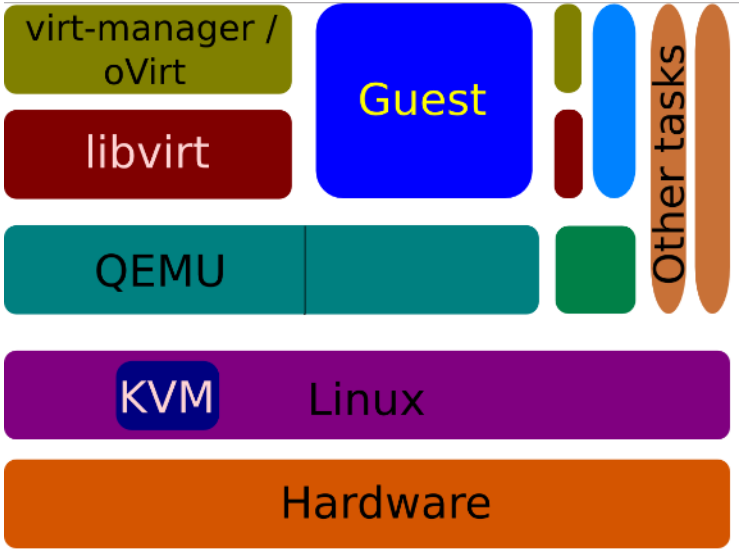
- **KVM's Killer Features**
  - For KVM, CPU is just a thread and VM is just a process, meaning that the hypervisor only runs the VM for you.
  - No policy decision is hypervisor's business, e.g. Security Checks and Memory Management and Scheduling done through Linux kernel
- **QEMU Crazy Features**
  - Virtual machine migration
  - SMP Symmetric Multi-Processing
  - Stable guest hardware. There will be no sudden changes in VM
  - Virtual machine monitor
  - KVM support

VM Management

- Creating VMs, from scratch or from templates
- Starting, suspending, and migrating VMs
- Using snapshots to back up and restore VMs
- Importing or exporting VMs
- Converting VMs from foreign hypervisors

Open Source VM Management

Virt-manager is a program used to manage VMs. It can talk to QEMU and KVM via LibVirt. In other words, LibVirt is the interface between QEMU and virt-manager



- **Libvirt**
  - Running our VM with QEMU is possible but it is not the perfect way
  - Libvirt is to provide a common and stable layer sufficient to securely manage domains on a node, possibly remote
  - In other words, it is an API that can be used both locally and remotely: to manage all components of our VM
  - It has a daemon too that:
    - Handle remote communication
    - Handle VM's lifecycle (start, stop, migrate, etc.)

- **LibVirt Supported Features**
  - **Comprehensive host management**
    - VMs
    - Virtual networks (bridging, NAT, etc.)
    - Storage (local, NFS, etc.)
  - **Isolation between VMs and Host**
    - Compromised VM can not access other VMs or hosts
    - Containment in case of hypervisor breaches
  - **Based on SElinux and cgroups**
    - SElinux: is a Linux kernel security module that provides a mechanism for supporting access control security policies.
    - Cgroups: is a Linux kernel feature that limits, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.

Large-scale VM Management

- **There is a Challenge to**
  - Manage ten of thousands of virtual machines
  - Each virtual machine potentially accessible from hundreds of nodes
- **Thus we need a management engine to take care of**
  - Creating VMs and provisioning them
  - Migrating them to another node
  - Load Balancing

Large-scale Virt. Solutions

- **OpenNebula, OpenStack compute (Nova)**
  - Used in Cloud deployment. Provisioning a cloud and letting our users run VMs on our cloud
- **Ganeti**
  - Ganeti is a virtual machine cluster management tool developed by Google. The solution stack uses either Xen, KVM as the virtualization platform.
- **Ovirt**
  - It provides a very complete administration interface where we can manage thousands of VMs on hundreds of nodes and assign storage to each VM, etc.

VM Migration

- **Migration Goals**
  - **Load balancing:** guests can be moved to hosts with lower usage when a host becomes overloaded
  - **Hardware failover Robustness:**, guests can be safely relocated so the host can be powered down and repaired, when hardware fails
  - **Energy Saving:** guests can be redistributed to other hosts and host systems powered off to save energy and cut cost in low usage periods.
  - **Geographic migration**: guests can be moved to another location for low latency or in serious circumstances

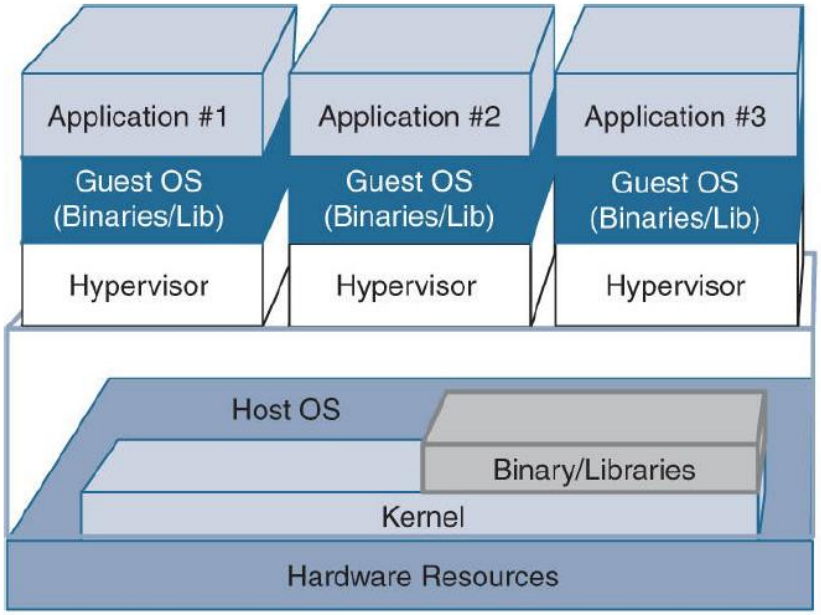
VM Migration

- **Migration Types**
  - **Offline Migration:** suspends the guest, then moves an image of the guest's memory to the destination host. The guest is resumed on the destination host and the memory the guest used on the source host is freed
  - **Live migration:** keeps the guest running on the source host, and begins moving the memory without stopping the guest
    - All modified memory pages are monitored for changes and sent to the destination while the image is sent.
    - The process continues until the amount of pause time allowed for the guest equals the predicted time for the final few pages to be transfer
    - The registers are loaded on the new host and the guest is then resumed on the destination host
    - If the guest cannot be merged (which happens when guests are under extreme loads) the guest is paused and then an offline migration is started instead

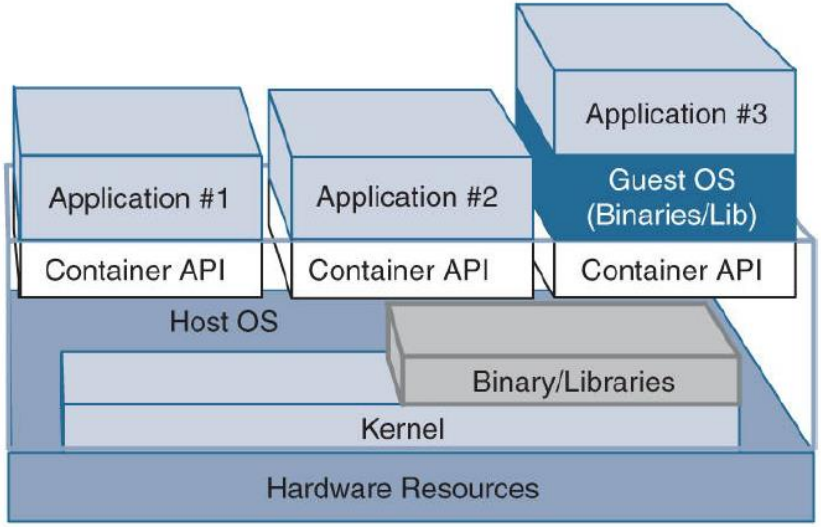
Container-based Virtualization

- Lightweight OS-level virtualization technique
- containers can run standalone application without any guest OS Providing kernel-level isolation; Linux Container (LXC)
- LXC is a set of functions and protocols for programs to interface with the application (also referred to as Application Programmable Interfaces or API).
- LXC APIs provide an interface to the container features offered by the Linux kernel.
- No portability supported
- Containerization vs. Virtualization

VM and Container Comparison



Virtual Machine



Linux Container

Containers vs. VMs

Virtual Machines	Containers
Allocate chunks of hardware resources to create a virtual machine	Process-level virtualization doesn't allocate hardware, but restricts usage and provides isolated view of hardware.
Needs an operating system with in the virtual machine	Can run standalone application (using the host OS as its operating system) or a different operating system (guest OS)
Can run any operating system (guest OS) on top of another (host OS)	The guest OS or application can only be something that is capable to run on the same kernel version that the host OS is running
Provides high level of isolation and security. Very unlikely for an application in the virtual machine to affect the other virtual machines or Host.	Doesn't provide pure isolation, as lots of components (including the kernel itself) are shared. Possible for one container's application to affect the entire host or other containers
Performance impact—usually slower since hardware emulation has to take place through the Hypervisor	Near native performance since there is no middle layer (virtualization layer, or Hypervisor)
Resource overhead—if the guest OS is running the same kernel as the host OS, then resources (disk space, memory, etc.) used by this guest are wasted.	No resource overhead, since the kernel's built-in capabilities are used.