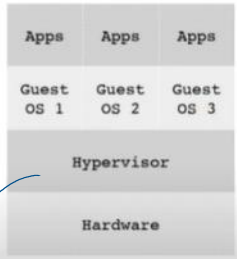
System VM

Hypervisors

- Also called Virtual Machine Monitor (VMM)
- A hypervisor is an operating system for operating systems
  - Provides a virtual execution environment for an entire OS and its applications
  - Controls access to hardware resources
  - When guest OS executes a privileged instruction, Hypervisor intercepts the instruction, checks for correctness and emulates the instruction.



VM's try to stay completely invisible

To do this they intercept instructions & emulates it.

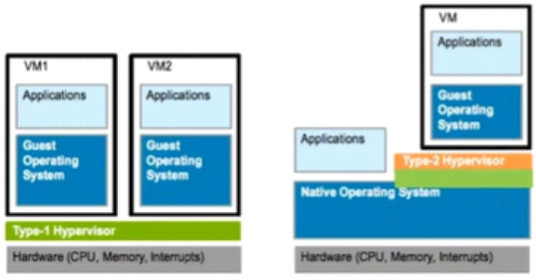
1st Hypervisor gets booted up

Guest OS does not know Hypervisor does this.

does same thing as OS but for VM's.

Type of Hypervisors

- Type 1 Hypervisor (bare metal, native): supports multiple virtual machines and runs directly on the hardware (e.g., VMware ESX , Xen, Denali)
- Type 2 Hypervisor (hosted) VM - runs under a host operating system (e.g., user-mode Linux)



https://microkerneldude.files.wordpress.com/2012/01/type1-vs-2.png

VM's will initiate & invoke previlleged instructions, But Hypervisior will trap them and emulate them.

⇒ Type-2 Hypervisior gives some flexibility

⇒ Type-2 Hypervisor gives some flexibility in terms of implementation.

⇒ we're making Hypervisor part of OS & user space.

⇒ it must've all drivers that is necessary to access hardware.

⇒ why'll write all drivers from scratch !

- already present in OS.
- why write which Type-1 Hypervisor doing.

⇒ we take advantage of already present drivers.

native/host OS — bare metal
guest OS — in VM running

Type-2 Hypervisors

Put Hypervisor in kernel module in OS to take over Previlleges from host OS & some lib. that are required to run apps.

Para-virtualized VMs

- Modify guest OS for better performance
- Traditional Hypervisors provide full-virtualization
  - They expose to VMs virtual hardware that is functionally identical to the underlying physical hardware.
  - Advantage : allows unmodified guest OS to execute
  - Disadvantage: Sensitive instructions must be trapped and emulated by Hypervisor.
  - E.g. KVM and VMWare ESX provide full virtualization

- Para-virtualized VM
  - Sees a virtual hardware abstraction that is similar, but not identical to the real hardware.
  - Guest OS is modified to replace sensitive instructions with "hypercalls" to the Hypervisor.
  - Advantage: Results in lower performance overhead
  - Disadvantage: Needs modification to the guest OS.
  - Xen provides both para-virtual as well as full-virtualization

→ if Hypervisor wants to get fully transparent all the instructs. must be trapped & emulated.

Para-virtualized VMs →

guest os modified
not all the instructions trapped & emulated

only sensitive info. } one set

which will be executed through hypervisor

& rest instr. will be system calls.

Hypervisor not fully transparent

→ often traditional hypervisors are partially Para-virtualized

↳ device drivers in guest OS may be Para-virtualized, whereas CPU & memory may be fully virtualized

Traditional Process vs traditional VMs

Resources shared among processes.

→ some virtualization/multiplexing done by OS in this level

→ loosely isolated

isolation b/w Process that are running on top of these

two guest OS is completely rigid.

⇒ resources

→ Process isolation

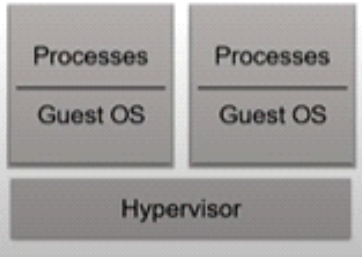
→ Processes will talk through system calls & eg. read() sys calls. in Linux.

→ all sys. calls available for all resources.

⇒ resources should also guest as we completely managed by Hypervisor

System VMs

- Each VM has its own
  - Guest OS
  - Guest physical memory ("virtualized" view of memory seen by guest OS)
  - One or more virtual CPUs
  - Virtual I/O devices: virtual disk, virtual network
- Ideally: Co-located VMs don't see/share ANYTHING

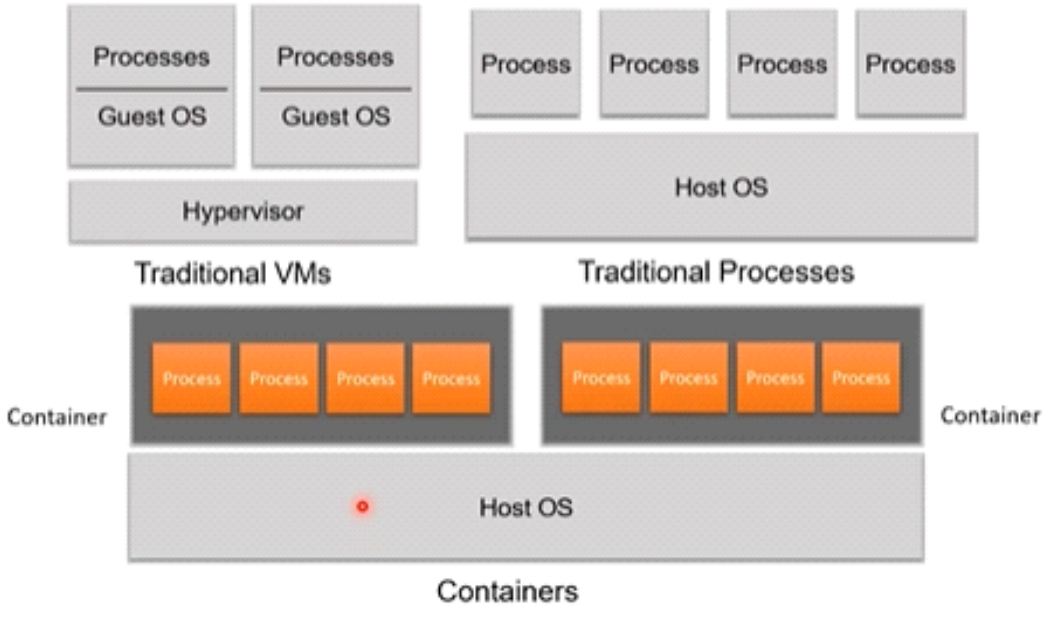


Isolation is important

- Limiting what/who a process/application can see.
- Limiting who can see a process/application

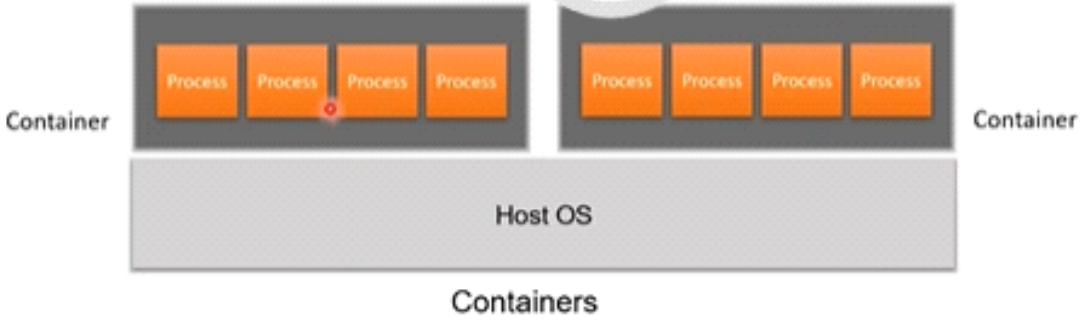
- Processes share too much
  - Great performance but not isolated enough
  - System VMs are too heavy
- Great Isolation but too heavy due to separate guest OS per VM Operating-system-level virtualization
  - Multiple isolated user-spaces
  - Share one kernel
  - Native performance

Process VMs and Containers



Containers

- Containers
  - group traditional processes together and
  - restrict what resources they can see/access.
- In Linux, containers consist of
  - Namespaces
  - Control Groups (cgroups)



Containers

Processes share namespaces, so if you create one specific namespace & control group of processes.

→ diff. containers will invoke host sys. calls to access OS.

→ in terms of sys level VMs & fully virtualized VMs — one cannot get full isolation of processes.

But with containers, we can get big reduction approach, while we can't've namespace separated

approach, where we can've namespaces separated for these processes & isolated resources sharing b/w these containers.

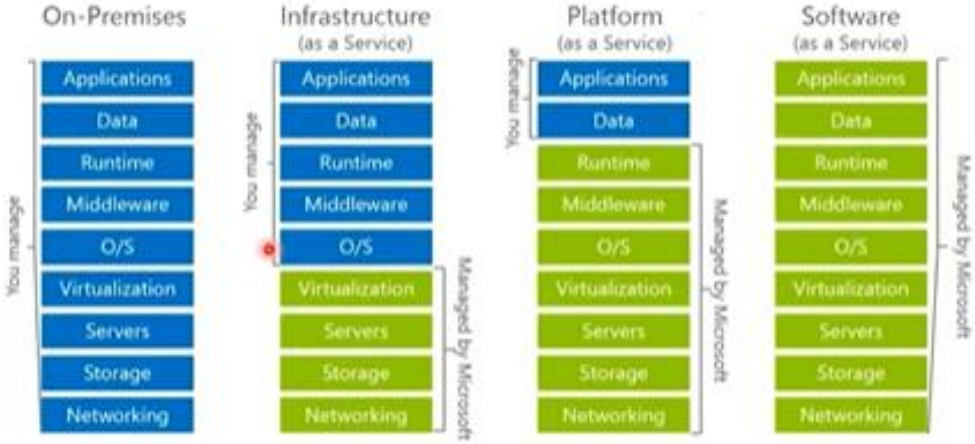
Cloud Computing

- Virtualized distributed processing, storage, and software resources and a service.
- Delivering computing as a on-demand, pay-as-you-go service.
- "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" – NIST definition

Service models

- IaaS: Infrastructure as a Service
  - Consumer can provision computing resources within provider's infrastructure upon which they can deploy and run arbitrary software, including OS and applications
- PaaS: Platform as a Service
  - Consumer can create custom applications using programming tools supported by the provider and deploy them onto the provider's cloud infrastructure
- SaaS: Software as a Service
  - Consumer uses provider's applications running on provider's cloud infrastructure

Service models



IAAS cloud acrhitecture