Virtualization

Don’t call it the cloud, just don’t
Outline

- Motivation
- Background
- Xen
- Containers
- Summary
Motivation

- Run multiple programs, and OSs, on a single system
- Need isolation between applications/OSs on the same machine
- Management (migration, snapshot, consolidation)
- Language VMs, code portability
Background - VMM Types

- Process vs system VMs
- Full system virtualization
  - No modification required for guest OS
  - Individual VMs may not even be aware of virtualization
- Paravirtualization
  - Slight modification required for guest OS, usually drivers
  - Individual VMs are aware they’re running in virtualized environment
Background - x86 CPU

- CPU Privilege levels (Protection rings)
- Allowed to execute different instructions depending on current state
- Kernel runs in “Ring 0”
- Applications in “Ring 3”
Xen

- Hypervisor-style system VMM, i.e. Xen runs on bare-metal
- Paravirtualized (at first)
- Minimal OS source modification (Linux, Windows XP)
- No change required for applications
- Low overhead
- Performance isolation
Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including Domain0 running control software in a XenoLinux environment.
Xen - CPU

• Xen VMM (hypervisor) runs in Ring 0

• Guest OS modified to run in Ring 1

• When privileged instruction is required, then trap to hypervisor

• Other hacks to avoid running in Ring 0 unless necessary, for performance reasons
Xen - Memory

• Hypervisor must also manage memory mapping for each guest

• Guests register and use their own page table with the MMU hardware, read-only
  • This avoids heavy overhead to access shadow page tables

• Only trap to Xen when modifications are required

• Balloon driver can be used to reduce host memory pressure
  • “Inflates” inside guest to acquire memory, then pass information to hypervisor for reclamation
Xen - Data Transfer

- Asynchronous I/O ring
- Contains *descriptors*, not the data!
- Xen and guest engaging in producer-consumer for I/O requests AND responses
- Re-ordering is allowed
- Notification is decoupled. Allowing latency and throughput trade-off

Figure 2: The structure of asynchronous I/O rings, which are used for data transfer between Xen and guest OSes.
Xen - Results

- Minimal overhead over native
- Strong performance isolation

Figure 3: Relative performance of native Linux (L), Xenolinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).
Modern Virtualization

- Much better hardware support: x86 Virtual Machine Extensions (VMX)
  - VMX root-mode, another set of privilege rings
  - Extended page tables
  - Device passthrough
Containers

- (Linux) Containers
- Process-level isolation
- Lower overhead compared to VM
- Same ISA, ABI as host OS for all containers
- Also often used for deployment
Containers

- Built on top of Linux subsystems
- *namespaces* for isolation (think *chroot*)
- *cgroups* for resource management
- LXC, Docker as management tools
Containers

- Containers can have single application, or collection of applications (good for deployment)
- Share the underlying Linux kernel
- Generally smaller footprint than full VM
- Easier to share resources between guests (good or bad?)
- Containers not aware of their resource limits (e.g. see all CPUs)
Containers

- Still under active development along with mainline Linux
  - e.g. Not all syscalls were namespace-aware, so root inside container = root outside container(!)

- AUFS is slow
- NAT is slow
Summary

• Do I need to know all the implementation details? Is this going to be on the exam?

• Motivation for virtualization, modern trends

• Hardware support? How have hardware and software features (co-)evolved?
  • x86 details also useful for everything else

• Differences and trade-offs between various virtualization/container technologies

• Basis for large section of the industry - Amazon EC2, Microsoft Azure, Google Compute Platform, Docker