Virtualization and Cloud Computing

Content taken from the following:
Various online sources
Problem

• Distributed systems are now ubiquitous
  – It’s hard to provide any software service at a modern scale from a single server
    • (Although if you can, you SHOULD!)
  – Most companies don’t need or want to manage their own hardware
    • High up-front costs, security vulnerabilities, etc.
  – Solution: abstraction!
    • In particular, abstracting away the hardware
      – Sometimes software too
    • Usually referred to as virtualization
Virtualization

- **Virtual environment**: abstract machine *(guest)* implemented on top of a physical machine *(host)*
  - Requires some kind of interpretation layer
- **Various goals**
  - **Emulation**: run programs designed for one architecture on another
  - **Isolation**: run programs in a sandbox
  - **Scalability**: spawn/destroy instances dynamically
  - **Automation**: reduce tedium and mistakes during deployment
  - **Reproducibility**: suspend/resume snapshots or configurations
Virtualization

• Various levels
  - Circuits / CPU (microcode emulating machine code)
  - Storage (e.g., RAID)
  - Networks (e.g., NAT or overlays)
  - Runtime environment (e.g., Java VM or Microsoft .NET)
  - Operating system (e.g., Docker)
  - Full desktops (e.g., QEMU, VMware or VirtualBox)
Hypervisors

- **Native** hypervisors (“type 1”)
  - Run directly on the host’s hardware in kernel mode
  - Sometimes as part of a general-purpose OS
  - Examples: VMware ESX, Microsoft Hyper-V, Oracle VM Server, Xen

- **Hosted** hypervisors (“type 2”)
  - Runs as a process inside the host OS
  - Often hardware-accelerated (e.g., Intel VT-x or AMD-V)
  - Examples: VMware Workstation, VirtualBox, QEMU
  - Sometimes referred to as an emulator if it virtualizes an entirely different architecture
    - Example: Project 4 in CS 261 is a Y86-64 emulator for x86-64
Windows: 3.1, 95, and 10 on 8.1

Image courtesy of Mike Ripley (JMU Infrastructure and Database Support)
**OS-level virtualization**

- **Container**: isolated user space for a program and its dependencies
  - Multiple user spaces implemented at the kernel level
  - Alternative viewpoints
    - Virtual memory extended to files and libraries
    - Sandbox, lightweight, app-specific VMs that run natively (no guest OS)
    - “Packages” for a single program's file system
  - **Portable**: code in the container will run the same everywhere
  - **Performant**: minimal overhead vs. running natively
  - Examples: chroot, FreeBSD jail, Docker
Cloud computing

- **Infrastructure-as-a-service (IaaS)**
  - Cloud provider owns the hardware (servers and NAS)
  - Clients provide virtual software images (VMware, Docker, etc.)
  - Inherent scalability (including dynamic provisioning) and fault-tolerance
  - Amazon EC2, Google Cloud, Microsoft Azure, Rackspace
Serverless computing

- Pay for compute time, not a particular host or VM
- FaaS: Function as a Service (another layer of abstraction!)
- There's still a server, but the user doesn't interact with it directly
- Code must be written using a supported language

https://aws.amazon.com/lambda/
Cloud engineering

- Emerging/developing field
  - Combines computer system engineering (EE), software engineering (CS), and computer information systems (business)
  - Focus on IaaS/PaaS/SaaS/FaaS applications
    - Often with a “big data” focus
  - Goals: performance, scalability, security, reliability
  - Challenge: integrating multiple solutions and layers
  - First IEEE International Conference on Cloud Engineering (IC2E) in March 2013
Thursday

- Cloud computing exercise
- Sign up for AWS account and apply for Educate credits:
  - http://aws.amazon.com/
  - https://aws.amazon.com/education/awssEducate/

Also: check your posters!