CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2024 L25

Virtualization and Data centers

Slides based on

- **Text by Silberschatz, Galvin, Gagne**
- **Various sources**

Project Notes

- **D3: Project Report Due 11/20/2024** Please see updated requirements. Slides should also be ready by 11/20/2024.
- Presentation schedule (12/2 to 12/5) will be posted later.
- **Project Slides** for both options need to be posted in Teams channel Project Slides (8- 10) and Videos 24 hours before schedule.
- Research Project **Videos** (7-8 min) should also be posted there by 24 hours before.
- Development Project **Demo schedule** (interactive using Teams) will be available later. Each team should sign up for one 15-min slot.

Project Notes: Peer Reviews

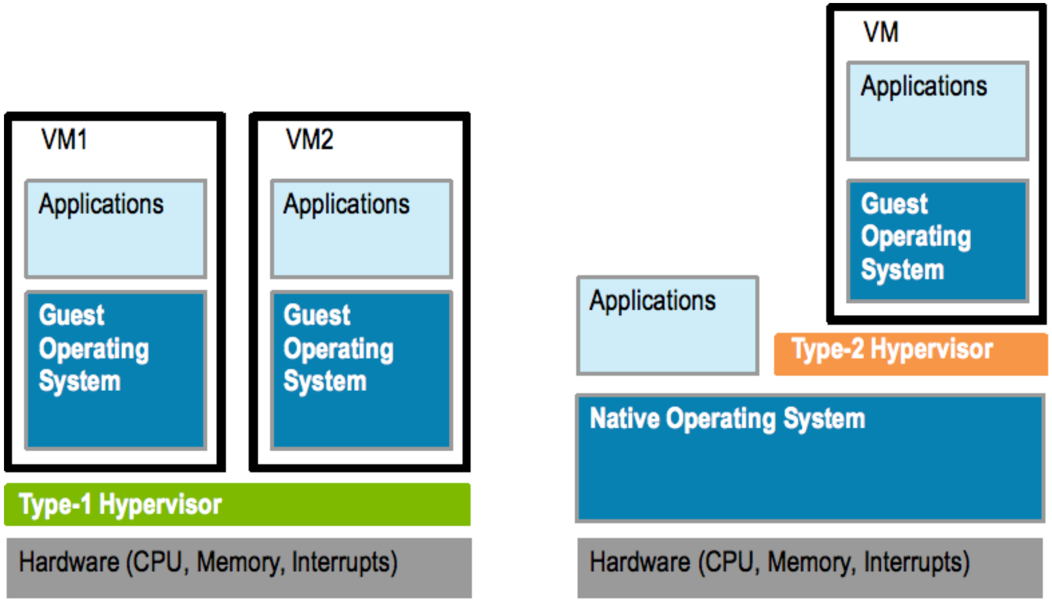
- Each student will need to view/evaluate
  - 2 assigned project reports
  - 7 videos/slides for A research projects (Sections 001, 801)
  - 3 videos/slides for B Development projects (Sections 001, 801)

  Use the review form and evaluation criteria that will be provided.

Some interesting courses

- CS435: Introduction to Big Data (Fall)
- CS456: Modern Cyber-Security
- CS470: Computer Architecture
- CS475: Parallel Programming/Processing (Spring)
- CS457: Computer Networks and the Internet
- CS530: Fault-Tolerant Computing (Spring)
- CS559: Quantitative Security (Fall)

Implementation of VMMs



What question do you see here?

- What mode does hypervisor run in? Guest OSs?
- Are Guest OSs aware of hypervisor?
- How is memory managed?
- How do we know what is the best choice? Answers coming up.

Colorado State University

Type 1 Hypervisors

- Run on top of *bare metal*
- Guest OSs believe they are running on bare metal, are unaware of hypervisor
  - are not modified
  - Better performance
- Choice for data centers
  - Consolidation of multiple OSes and apps onto less HW
  - Move guests between systems to balance performance
  - Snapshots and cloning
- Hypervisor creates runs and manages guest OSes
  - Run in kernel mode
  - Implement device drivers
  - provide traditional OS services like CPU and memory management
- Examples: VMWare esx (dedicated) , Windows with Hyper-V (includes OS)

Type 2 Hypervisors

- Run on top of host OS
- VMM is simply a process, managed by host OS
  - host doesn't know they are a VMM running guests
- poorer overall performance because can't take advantage of some HW features
- Host OS is just a regular one
  - could have Type 2 hypervisor (e.g. Virtualbox) on native host (perhaps windows), run one or more guests (perhaps Linux, MacOS)

Full vs Para-virtualization

- Full virtualization: Guest OS is unaware of the hypervisor. It thinks it is running on bare metal.
- Para-virtualization: Guest OS is modified and optimized. It sees underlying hypervisor.
  - Introduced and developed by Xen
    - Modifications needed: Linux 1.36%, XP: 0.04% of code base
  - Does not need as much hardware support
  - allowed virtualization of older x86 CPUs without binary translation
  - Not used by Xen on newer processors

CPU Scheduling

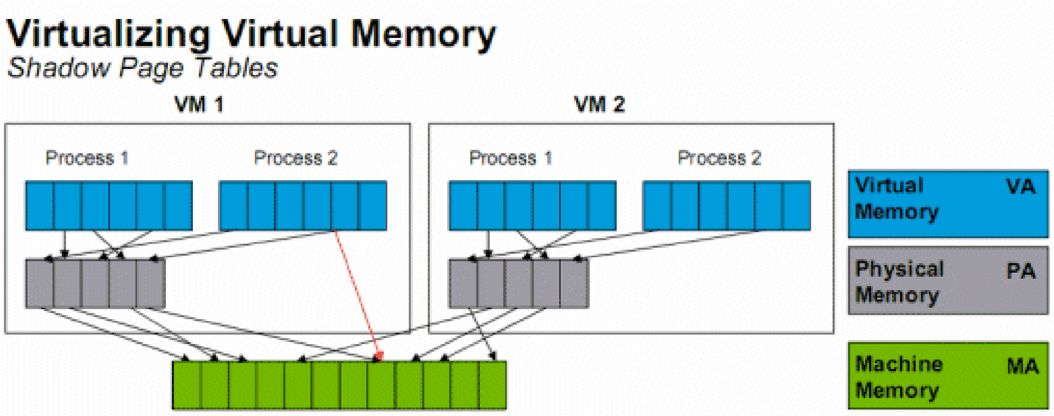
- One or more virtual CPUs (vCPUs) per guest
  - Can be adjusted throughout life of VM
- When enough CPUs for all guests
  - VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs (CPU overcommitment)
  - VMM can use scheduling algorithms to allocate vCPUs
  - Some add fairness aspect
- Oversubscription of CPUs means guests may not get CPU cycles they expect
  - Time-of-day clocks may be incorrect
  - Some VMMs provide application to run in each guest to fix time-of-day

Memory Management

Memory mapping:

- On a bare metal machine: OS uses page table/TLB to map Virtual page number (VPN) to Physical page number (PPN) (physical memory is shared). Each process has its own page table/TLB.
  - VPN -> PPN
- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

  PPN ->MPN



Memory Management

- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

  PPN ->MPN

- Where is this done?
  - Has to be done by hypervisor $_{\text{type 1}}$. Guest OS knows nothing about MPN.
  - Page Table/TLB updates are trapped to VMM.

    It needs to do VPN->PPN ->MPN.
  - It can do VPN->MPN directly (VMware ESX)

Virtual Machine (VM) as a software construct

- Each VM is configured with some number of processors, some amount of RAM, storage resources, and connectivity through the network ports.
- Once the VM is created it can be activated on like a physical server, loaded with an operating system and software solutions, and used just like a physical server.
- Unlike a physical server, VM only sees the resources it has been configured with, not all of the resources of the physical host itself.
- The hypervisor facilitates the translation and I/O between the virtual machine and the physical server.

Virtual Machine (VM) as a set of files

- Configuration file describes the attributes of the virtual machine containing
  - server definition,
  - how many virtual processors (vCPUs)
  - how much RAM is allocated,
  - which I/O devices the VM has access to,
  - how many network interface cards (NICs) are in the virtual server
  - the storage that the VM can access
- When a virtual machine is instantiated, additional files are created for logging, for memory paging etc.
- Copying a VM produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself

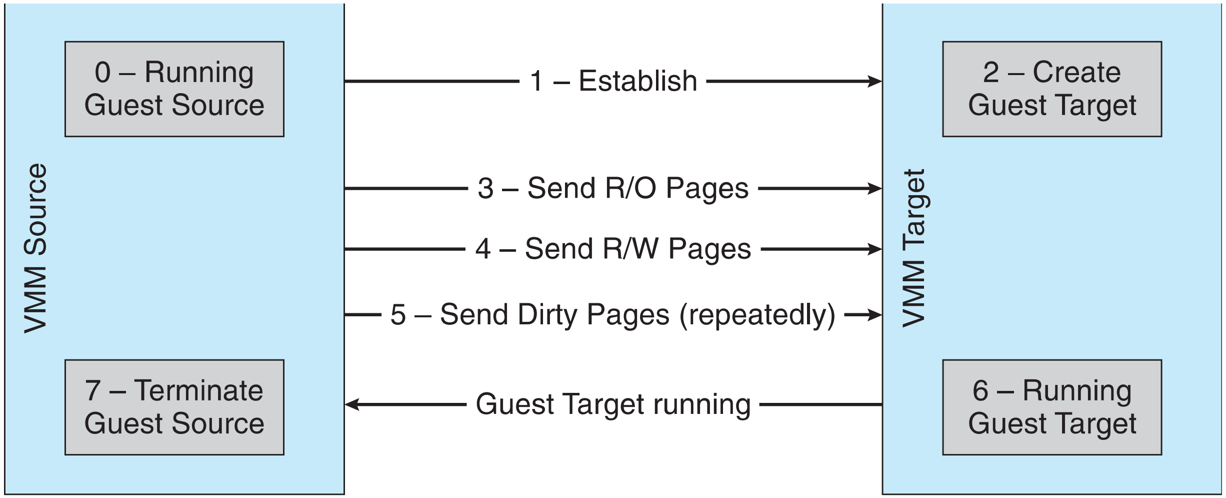
Live Migration

Running guest can be moved between systems, without interrupting user access to the guest or its apps

- for resource management,
- maintenance downtime windows, etc

- Migration from source VMM to target VMM
  - Needs to migrate all pages gradually, without interrupting execution (details in next slide)
  - Eventually source VMM freezes guest, sends vCPU's final state, sends other state details, and tells target to start running the guest
  - Once target acknowledges that guest running, source terminates guest

Live Migration



- Migration from source VMM to target VMM
  - Source establishes a connection with the target
  - Target creates a new guest
  - Source sends all read-only memory pages to target
  - Source starts sending all read-write pages
  - Source VMM freezes guest, sends final stuff,
  - Once target acknowledge that guest running, source terminates guest.

VIRTUAL APPLIANCES: “shrink-wrapped” virtual machines

- Developer can construct a virtual machine with
  - required OS, compiler, libraries, and application code
  - Freeze them as a unit … ready to run
- Customers get a complete working package
- Virtual appliances: “shrink-wrapped” virtual machines
- Amazon’s EC2 cloud offers many pre-packaged virtual appliances examples of *Software as a service*

- *Question: do we really have to include a whole kernel in a shrink wrapped VM?*

CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Back from ICQ

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2024

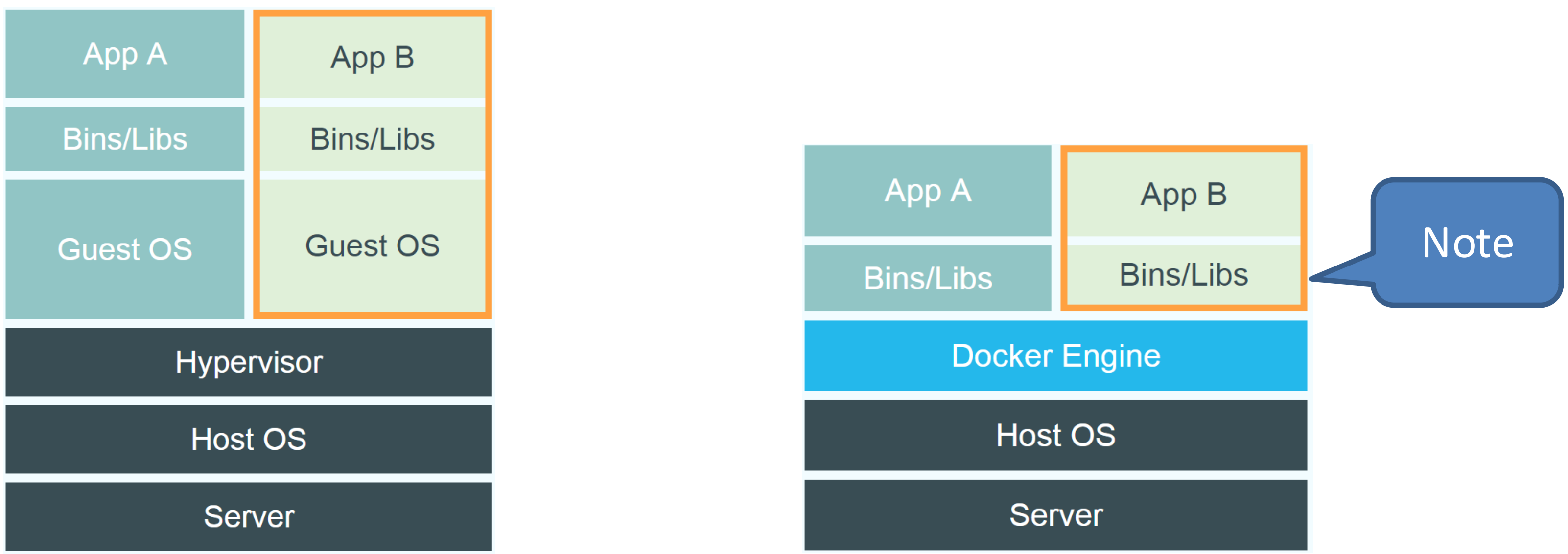
Containers

Slides based on

- Various sources

Linux Containers and Docker

- Linux containers (LXC 2008) are “lightweight” VMs
- Comparison between LXC/docker (2013) and VM



- Containers provide “OS-level Virtualization” vs “hardware level”.
- Containers can be deployed in seconds.
- Very little overhead during execution, even better than Type 1 VMM.

VMs vs Containers

VMs	Containers ("virtual environment")
Heavyweight several GB	Lightweight tens of MB
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	*OS virtualization*
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

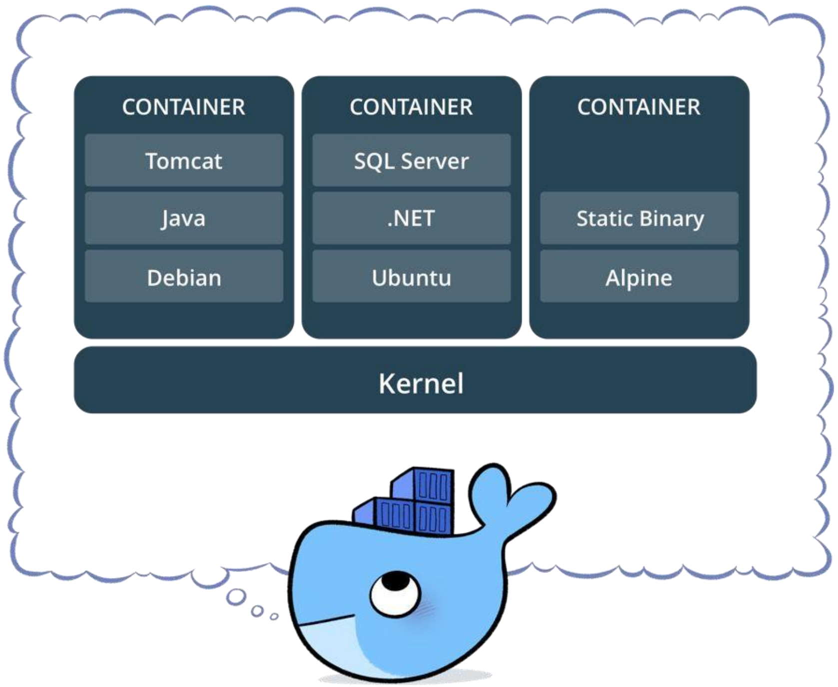
Container: basis

Linux kernel provides

- "control groups" (cgroups) functionality for a set of processes
  - allows allocation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any VM
- "namespace isolation" functionality
  - allows complete isolation of an applications' view of the operating environment including Process trees, networking, user IDs and mounted file systems.
- Managed by
  - Docker (or competitors) Platform: build, share, run containerized apps.
  - Kubernetes (or competitors): orchestration platform for managing, automating, and scaling containerized applications

Docker – podman/buildah   Docker swarm – Kubernetes, OPENSHIFT

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Docker Desktop for Windows uses Windows-native Hyper-V virtualization (Win10)
- Containers native to Windows Server 2016
- Docker: a popular container management service technology.

  Alternatives: Podman etc

Some Docker vocabulary

- **Docker Image**
  - The basis of a Docker container. Represents a full application
- **Docker Container**
  - The standard unit in which the application service resides and executes
- **Docker Engine**
  - Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider
- **Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))**
  - Cloud or server based storage and distribution service for images (can be **pull**ed or **push**ed)
- **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image using **docker build** command.

Correspondence: excecutable code:image container:process

Some Docker vocabulary: Analogies

Containers have their own jargon. Here are some analogous terms. Note that some analogies can be questionable.

	Docker	Non-containerized code
What is executed	Docker Image	executable
Isolation unit	Docker Container	process
to create what is executed	Dockerfile	makefile
	Docker engine	OS/JVM
	Registry Service	code repository

- Only a high-level look here. For details see documentation and videos.
- Several interrelated technologies. Significant experience needed to gain expertise.

Some Docker vocabulary

- **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image using **docker build** command.
- Ex:

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Each instruction creates one layer:

- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client's current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.

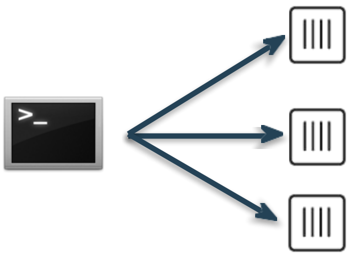
Docker Volumes

- Volumes mount a directory on the host into the container at a specific location
- Can be used to share (and persist) data between containers
  - Directory persists after the container is deleted
    - Unless you explicitly delete it
- Can be created in a Dockerfile or via CLI

Docker Compose: Multi Container Applications

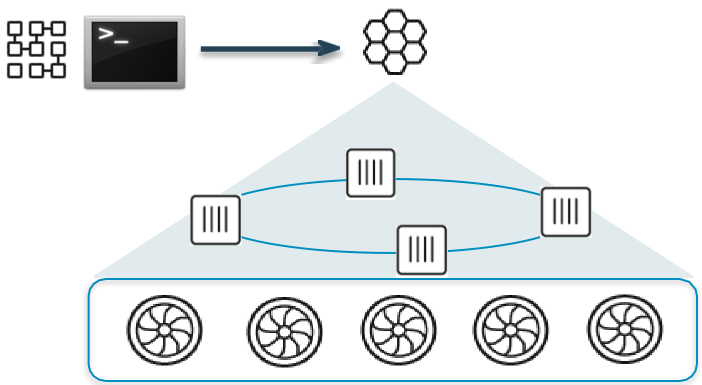
Single container

- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



Multi-container application

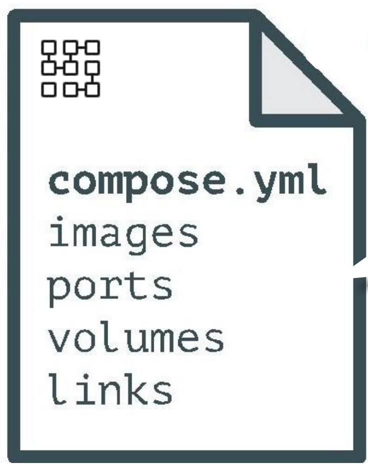
- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane



49

Colorado State University

Docker Compose: Multi Container Applications




```
version: '2' # specify docker-compose version

# Define the services/containers to be run
services:
angular: # name of the first service
build: client # specify the directory of the Dockerfile
ports:
- "4200:4200" # specify port forewarding

express: #name of the second service
build: api # specify the directory of the Dockerfile
ports:
- "3977:3977" #specify ports forewarding

database: # name of the third service
image: mongo # specify image to build container from
ports:
- "27017:27017" # specify port forewarding
```

Colorado State University

Terms

- **Docker** technology used for containers and can deploy single, containerized applications.
- **Docker Compose** for configuring and starting multiple Docker containers on the same host.
- **Docker swarm** is a container orchestration tool that allows you to run and connect containers on multiple hosts.
- **Kubernetes** is a container orchestration tool that is similar to Docker swarm, but has ease of automation and ability to handle higher demand.

Some Docker Commands

- **docker — — version** get the currently installed version of docker
- **docker build <path to docker file>** build an image from a specified docker file
- **docker login** login to the docker hub repository

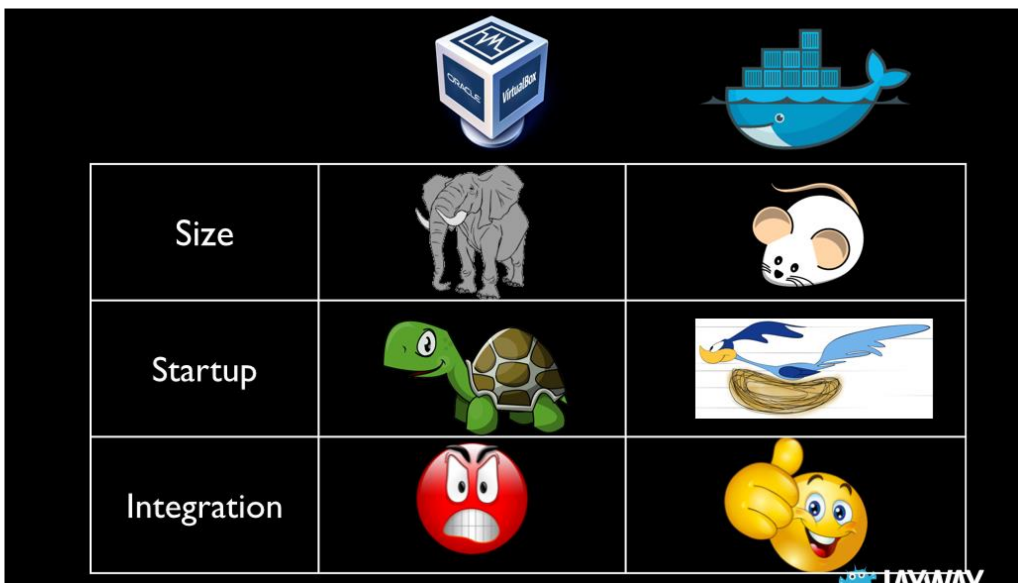
- **docker pull <image name>** pull images from the **docker repository** hub.docker.com
- **docker push <username/image name>**

- **docker run -it -d <image name>** create a container from an image
- **docker stop <container id>** stops a running container
- **docker kill <container id>** kills the container by stopping its execution immediately
- **docker rm <container id>** delete a stopped container
- **docker ps** list the running containers

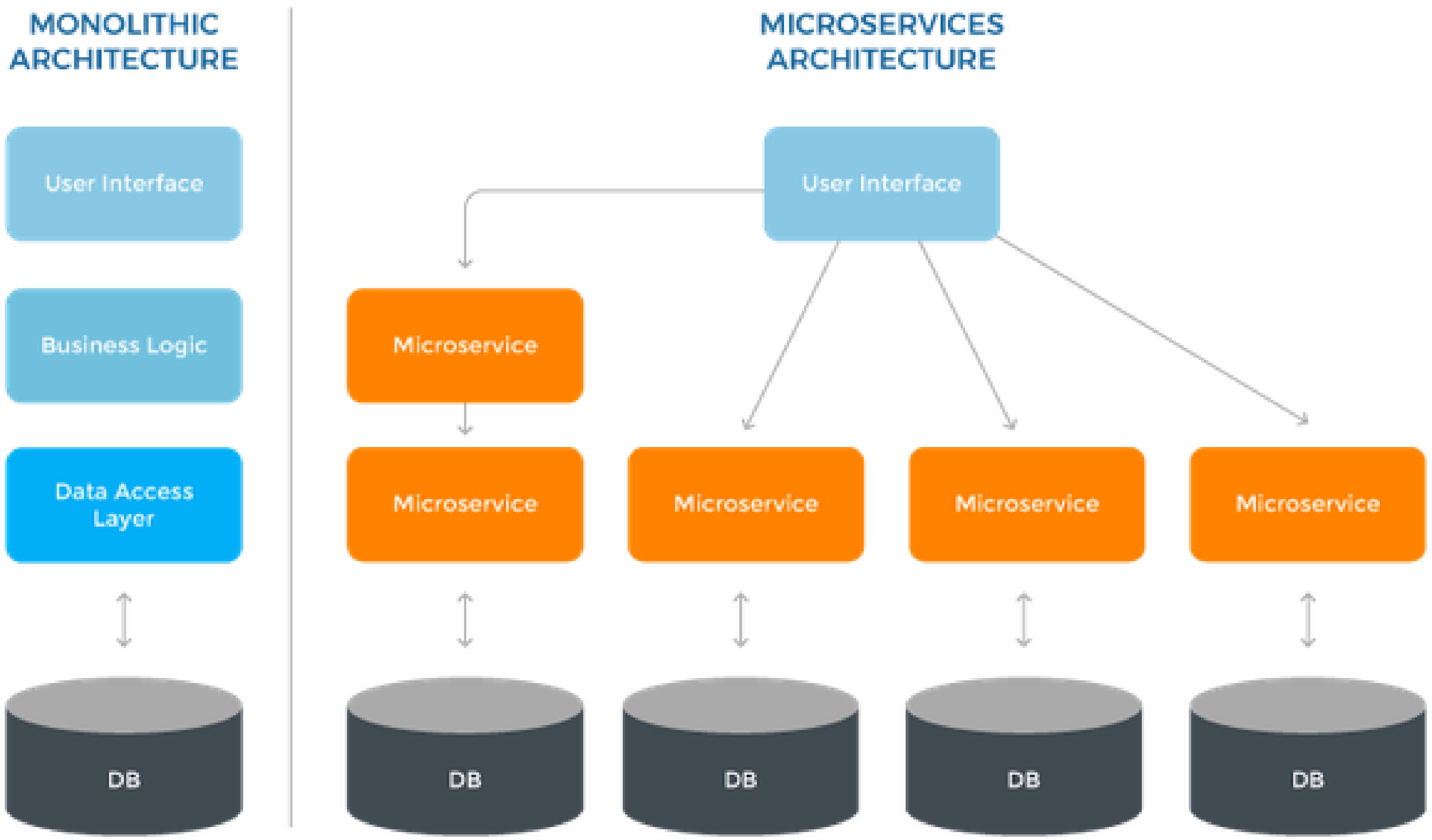
- **docker exec -it <container id> bash** to access the running container
- **docker commit <conatainer id> <username/imagename>** creates a new image of an edited container
- **docker images** lists all the locally stored docker images

Unique features

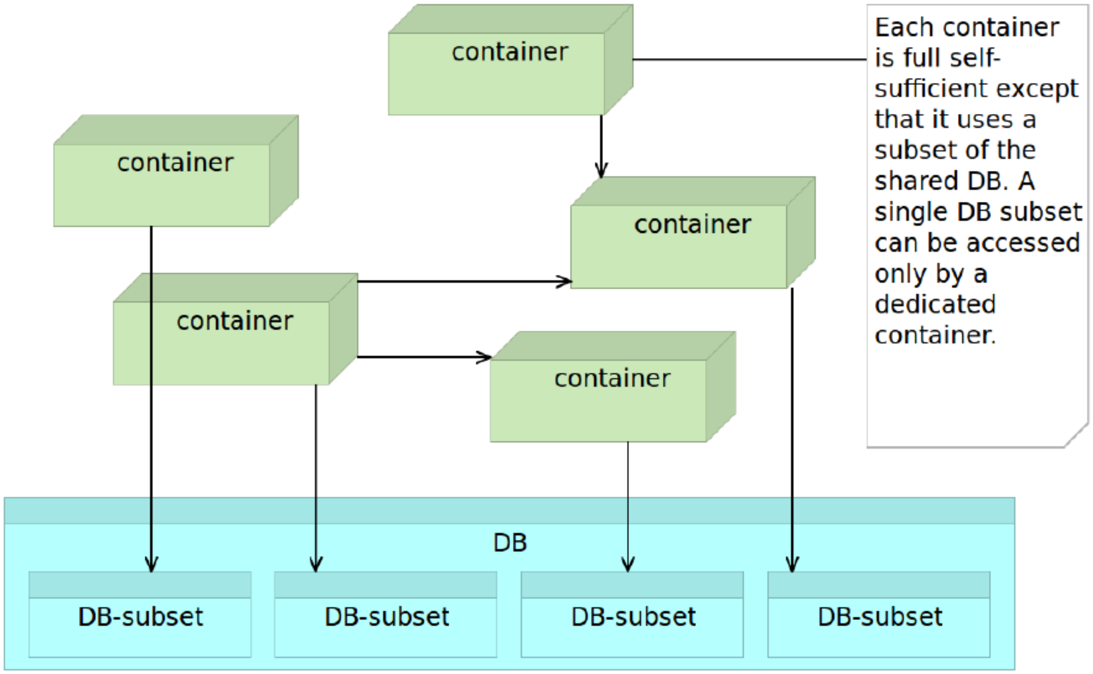
- Containers run in the user space
- Each container has its own: process space, network interface, booting mechanism with configuration
- Share kernel with the host
- Can be packaged as Docker images to provide *microservices*.



Monolithic architecture vs microservices



Microservices Accessing the Shared Database

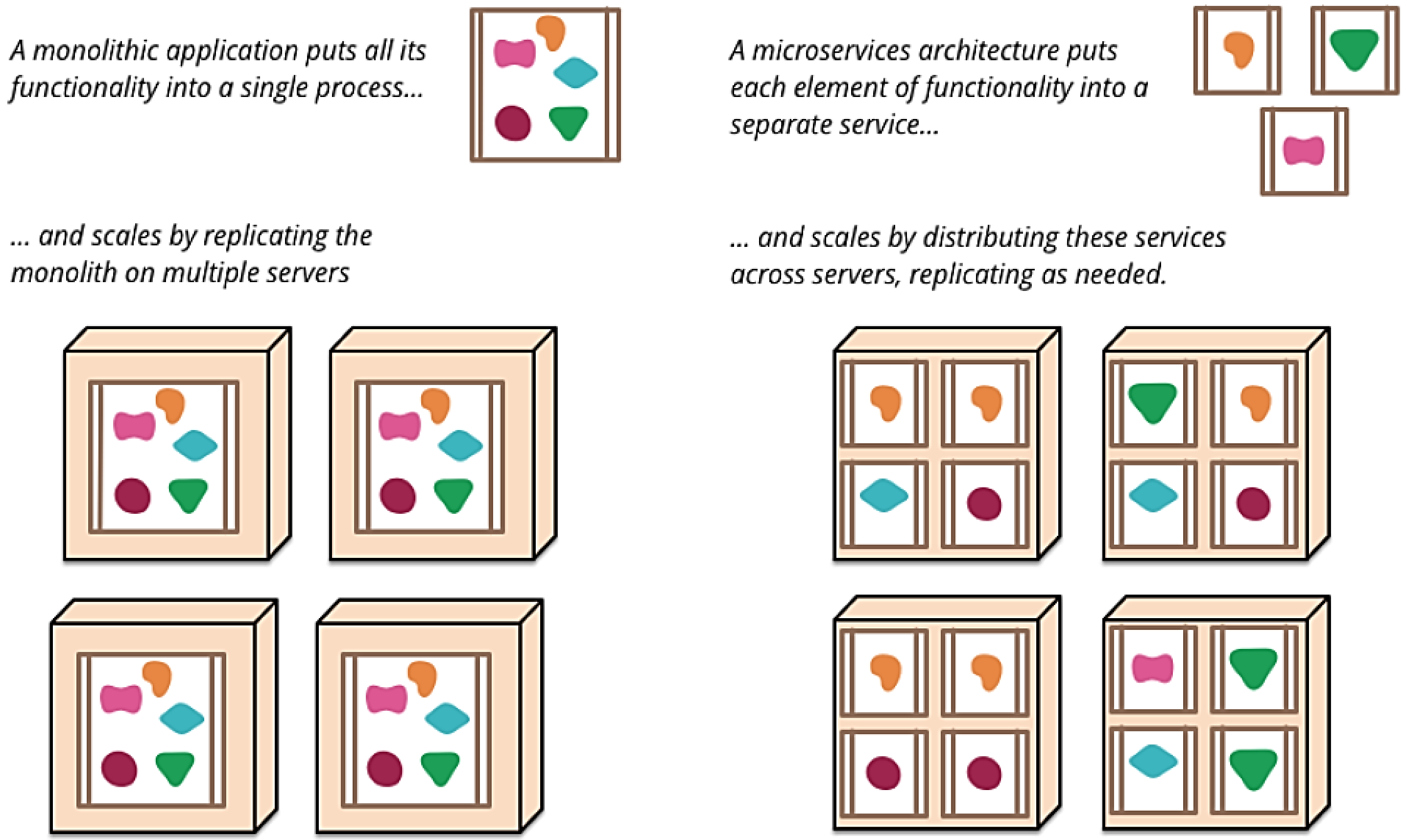


Microservices Characteristics

- Many smaller (fine grained), clearly scoped services
  - Single Responsibility Principle
  - Independently Managed
- Clear ownership for each service
  - Typically need/adopt the "DevOps" model
- 100s of MicroServices
  - Need a Service Metadata Registry (Discovery Service)
- May be replicated as needed
- A microservice can be updated without interruption

Colorado State University

Microservices. Scalability



CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2024

Data Centers & Cloud Computing

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Data Centers

- Large server and storage farms
  - 1000s-100,000 of servers
  - Many PBs of data
- Used by
  - Enterprises for server applications
  - Internet companies
  - Some of the biggest DCs are owned by Google, Facebook, etc
- Used for
  - Data processing
  - Web sites
  - Business apps

Data Center architecture

Traditional - static

- Applications run on physical servers
- System administrators monitor and manually manage servers
- Storage Array Networks (SAN) or Network Attached Storage (NAS) to hold data

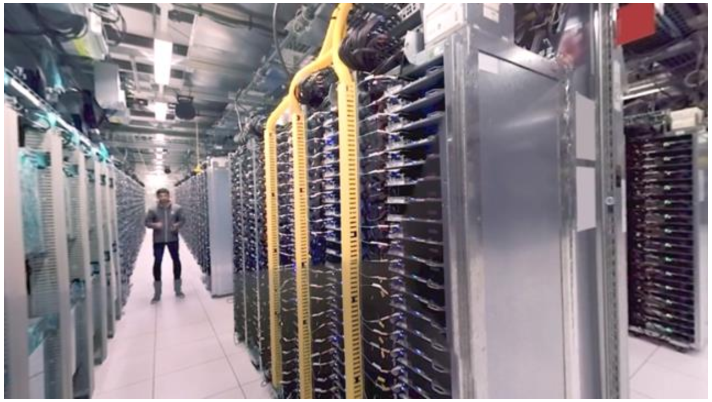
Modern – dynamic with larger scale

- Run applications inside virtual machines
- Flexible mapping from virtual to physical resources
- Increased automation, larger scale

Data Center architecture

Giant warehouses with:

- Racks of servers
- Storage arrays
- Cooling infrastructure
- Power converters
- Backup generators



Or with containers

- Each container filled with thousands of servers
- Can easily add new containers
- “Plug and play”
- Pre-assembled, cheaper, easily expanded

Server Virtualization

Allows a server to be "sliced" into Virtual Machines

- VM has own OS/applications
- Rapidly adjust resource allocations
- VM migration within a LAN

- Virtual Servers
  - Consolidate servers
  - Faster deployment
  - Easier maintenance
- Virtual Desktops
  - Host employee desktops in VMs
  - Remote access with *thin clients*
  - Desktop is available anywhere
  - • Easier to manage and maintain

Data Center Challenges

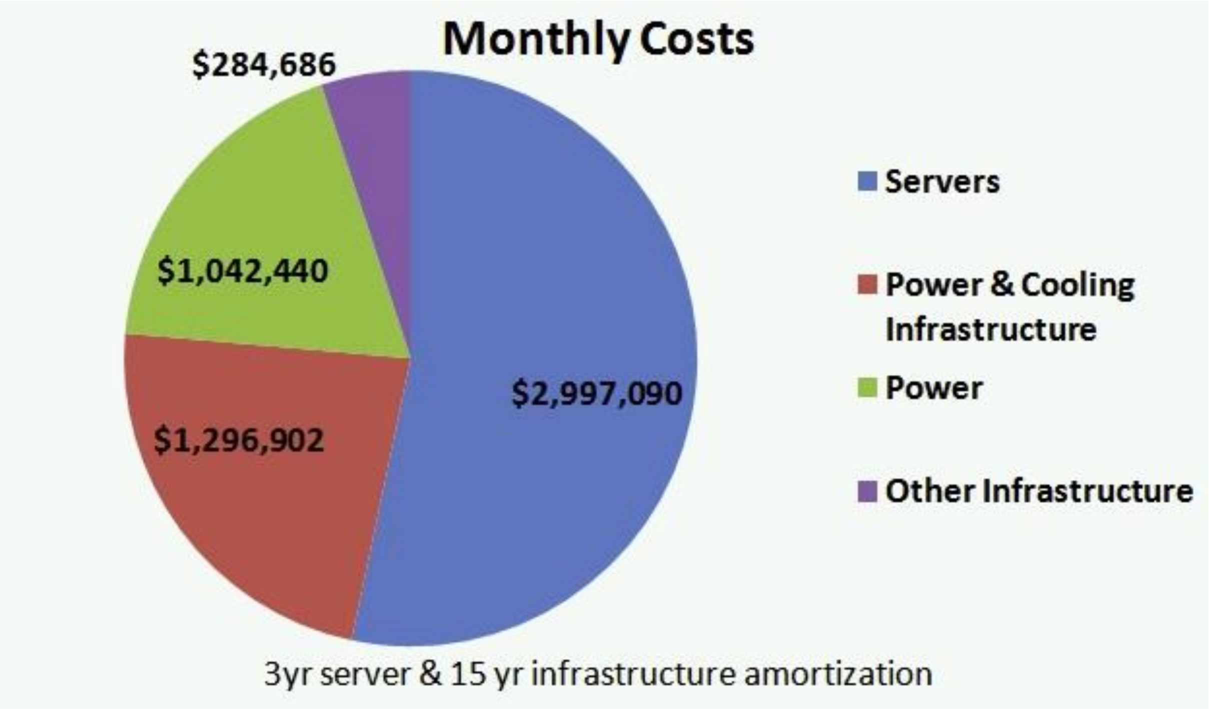
Resource management

- How to efficiently use server and storage resources?
- Many apps have variable, unpredictable workloads
- Want high performance and low cost
- Automated resource management
- Performance profiling and prediction

Energy Efficiency

- Servers consume huge amounts of energy
- Want to be "green"
- Want to save money

Data Center Challenges



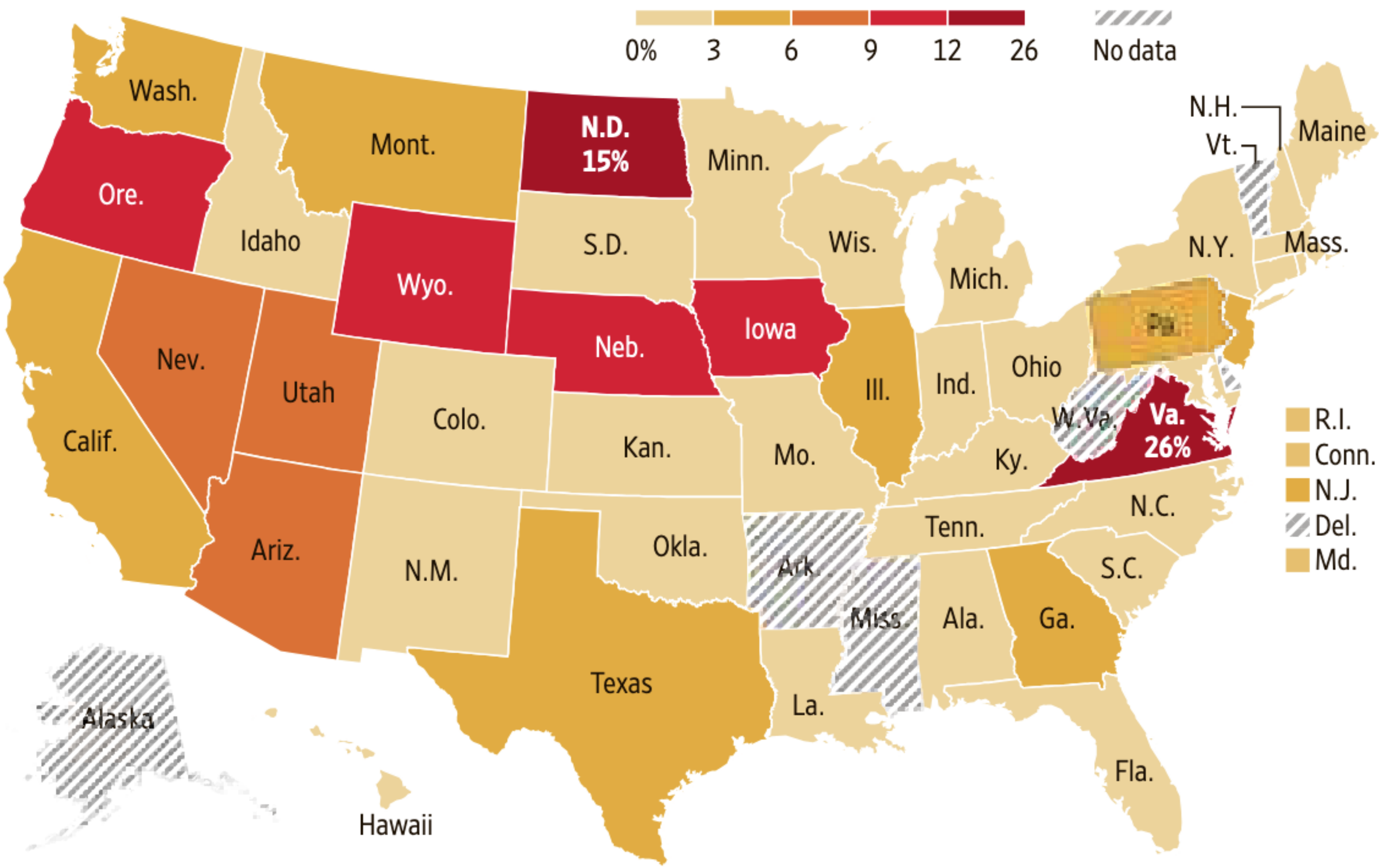
Power Efficiency measured as *Power Usage Effectiveness*

- *Power Usage Effectiveness* = Total Power / IT Power
- typical: 1.7, Google PUE ~ 1.1)

http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx

Power Consumption by Data Centers

Data centers' share of total power consumption, 2023



Source: Electric Power Research Institute

Colorado State University

Economy of Scale

Larger data centers can be cheaper to buy and run than smaller ones

- Lower prices for buying equipment in bulk
- Cheaper energy rates
- Automation allows small number of sys admins to manage thousands of servers
- General trend is towards larger mega data centers
- 100,000s of servers
- Has helped grow the popularity of cloud computing

Economy of Scale

Resource	Cost in Medium DC	Cost in Very Large DC	Ratio
CPU cycle cost	2 picocents	< 0.5 picocents	
Network	$95 / Mbps / month	$13 / Mbps / month	7.1x
Storage	$2.20 / GB / month	$0.40 / GB / month	5.7x
Administration	≈140 servers/admin	>1000 servers/admin	7.1x

Pico = 10^{-3} nano = 10^{-12}

Reliability Challenges

Typical failures in a year of a Google data center:

- 20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- 3 router failures (have to immediately pull traffic for an hour)
- 1000 individual machine failures
- thousands of hard drive failures

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/people/jeff/stanford-295-talk.pdf

Capacity provisioning

User has a variable need for capacity. User can choose among

Fixed resources: Private data center

- Under-provisioning when demand is too high, or
- Provisioning for peak

Variable resources:

- Use more or less depending on demand
- Public Cloud has elastic capacity (i.e. way more than what the user needs)
- User can get exactly the capacity from the Cloud that is actually needed

Why does this work for the provider?

  - Varying demand is statistically smoothed out over many users, their peaks may occur at different times
  - Prices set low for low overall demand periods

Amazon EC2 Instance types

On-Demand instances

- Users that prefer the low cost and flexibility of Amazon EC2 without any up-front payment or long-term commitment
- Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted

Spot Instances (cheap)

- request spare Amazon EC2 computing capacity for up to 90% off
- Applications that have flexible start and end times

Reserved Instances (expensive)

- Applications with steady state usage
- Applications that may require reserved capacity

Dedicated Hosts

- physical EC2 server dedicated for your use.
- server-bound software licenses, or meet compliance requirements

Amazon EC2 Prices (samples from their site)

General Purpose - Current Generation Region: US East (Ohio)

instance	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
t2.nano	1	Variable	0.5	EBS Only	$0.0058 per Hour
t2.small	1	Variable	2	EBS Only	$0.023 per Hour
t2.medium	2	Variable	4	EBS Only	$0.0464 per Hour
m5.4xlarge	16	61	64	EBS Only	$0.768 per Hour
m4.16xlarge	64	188	256	EBS Only	$3.2 per Hour

ECU = EC2 Compute Unit (perf), EBS: elastic block store (storage) , automatically replicated

Host OS answer

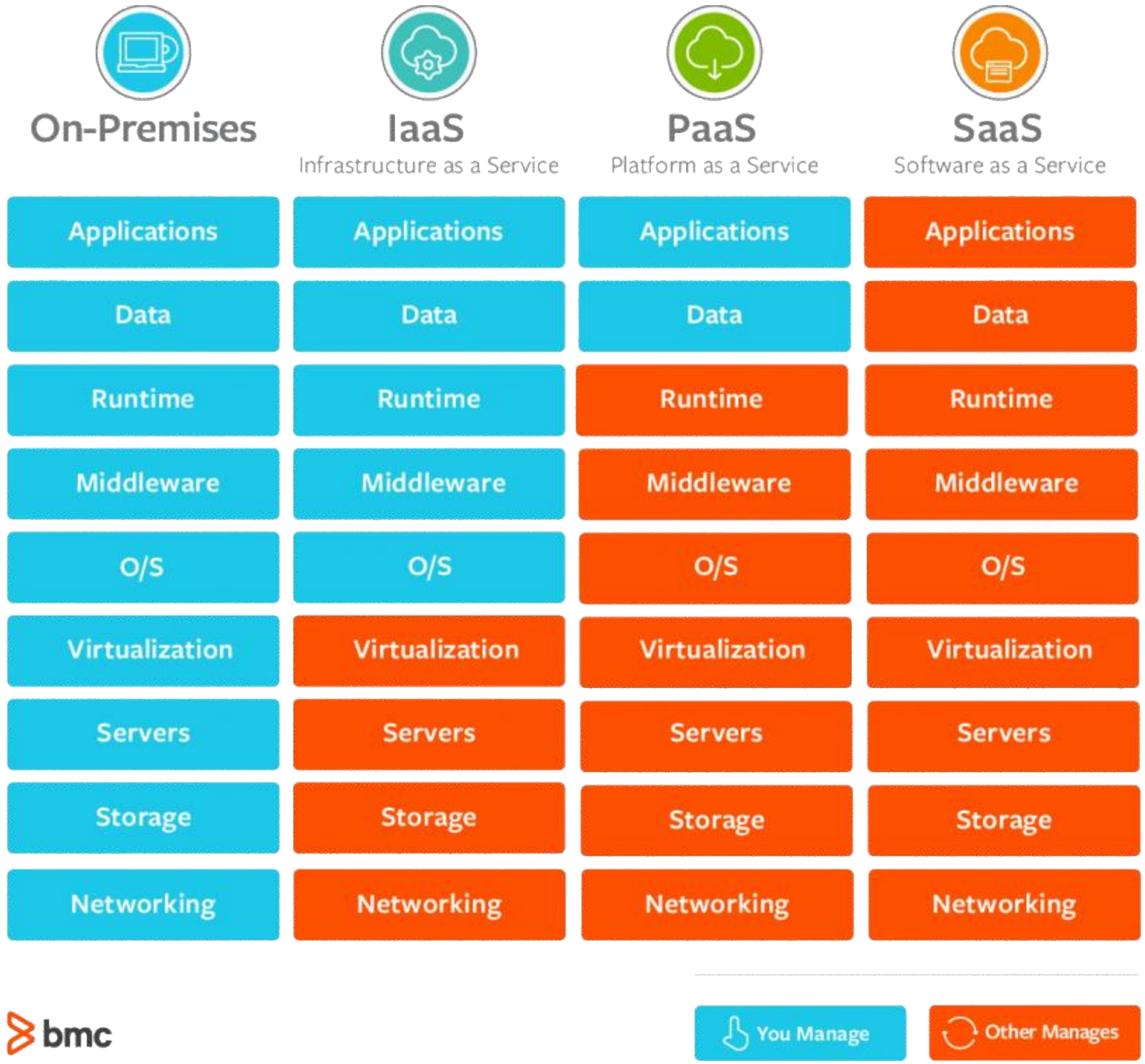
1. **In Type 1 VMM, is there a host OS?** **No.** Hypervisor services the guest Oss.

2. **Can a single hypervisor manage VMs with different OSs, win, linux, MacOS?** **Yes**

Service models

- IaaS: Infrastructure as a Service
  - infrastructure components traditionally present in an on-premises data center, including servers, storage and networking hardware
  - e.g., Amazon EC2, Microsoft Azure, Google Compute Engine
- PaaS: Platform as a Service
  - supplies an environment on which users can install applications and data sets
  - e.g., Google AppEngine, Heroku, Apache Stratos
- SaaS: Software as a Service
  - a software distribution model with provider hosted applications
  - Microsoft Office365, Amazon DynamoDB, Gmail

The Service Models



https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/

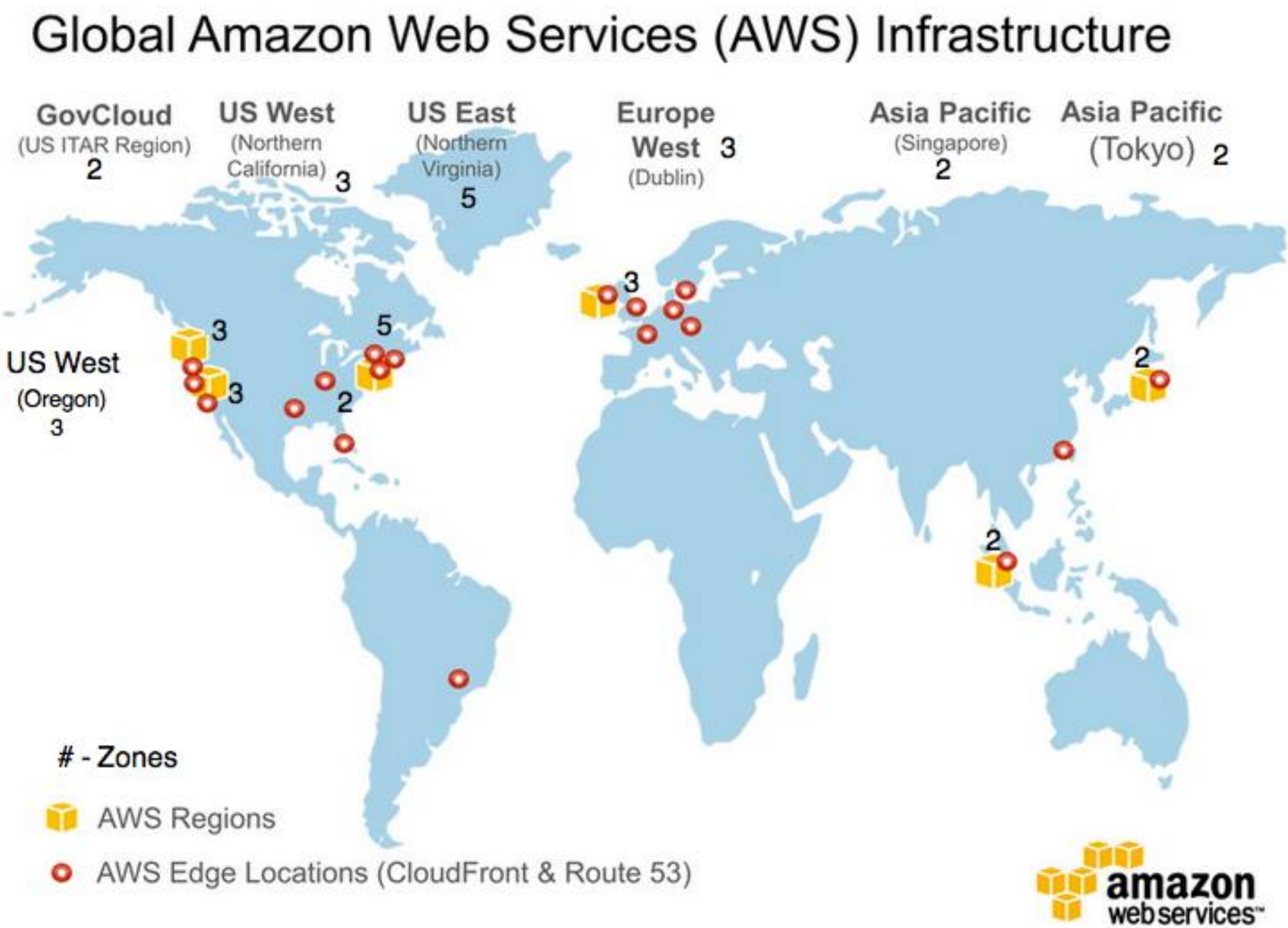
Cloud Management models

- **Public clouds**
  - Utility model
  - Shared hardware, no control of hardware,
  - Self-managed (e.g., AWS, Azure)
- **Private clouds**:
  - More isolated (secure?)
  - Federal compliance friendly
  - Customizable hardware and hardware sharing
- **Hybrid clouds**:
  - a mix of on-premises, private cloud and third-party, public cloud services.
  - Allows workloads to move between private and public clouds as computing needs and costs change.

Different Regions to Achieve HA

- AWS datacenters is divided into regions and zones,
  - that aid in achieving availability and disaster recovery capability.
- Provide option to create point-in-time snapshots to back up and restore data to achieve DR capabilities.
- The snapshot copy feature allows you to copy data to a different AWS region.
  - This is very helpful if your current region is unreachable or there is a need to create an instance in another region
  - You can then make your application highly available by setting the failover to another region.

Different Regions to Achieve HA



Colorado State University

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Spring 2022

Security

Slides based on

- **Various sources**

Security System Architecture

- Networked systems
  - Use of firewalls: Organization wide and system level
  - Address translation
  - Isolation of systems
- Single computing System: OS
  - Multiple levels of priviledges
  - Isolation of
    - processes,
    - cgroups,
    - virtual machines