Distributed Systems

Virtualization in Distributed Environments

WEEK 4: Virtualization, Virtual Machines, Containers, Resource Isolation, Cloud Infrastructure.

Online Lecture Series - 4

Felix Edesa

Addis Ababa Science and Technology University

AASTU

Course Outline

Topics We Will Cover

- **Virtualization** — Principles, Types, Hypervisors
- **Virtual Machines (VMs)** — Components, Lifecycle, Benefits, Examples
- **Containers** — Docker, Kubernetes, Images, Runtime, Benefits
- **Resource Isolation** — CPU, Memory, Disk, Network, Security Stability
- **Cloud Infrastructure** — IaaS/PaaS/SaaS, Multi-Tier Apps, Real-World Use

AASTU

Implementation Levels of Virtualization

What is Virtualization?

- Technology allowing multiple **virtual machines (VMs)** to share one hardware system.
- Originated in the 1960s, revitalized by cloud and distributed computing.
- Purpose: **resource sharing**, improved utilization, and application flexibility.
- Hardware (CPU, memory, I/O) and software (OS, libraries) can be virtualized.
- Separates hardware from software for **better efficiency**.

AASTU

Virtualization in Distributed Systems

Why Virtualization Matters?

- Provides **large memory space** and resource abstraction (e.g., virtual memory).
- Enables efficient use of **compute engines, storage, and networks**.
- Applications and operating systems can run independently on shared hardware.
- 2009 Gartner Report: virtualization was the **top strategic technology**.

AASTU

Virtualization Architecture

Traditional vs Virtualized Systems

- **Traditional System:** Host OS directly manages hardware.
- **Virtualized System:** A virtualization layer (**hypervisor / VMM**) is added.
- Guest OS runs on virtualized CPU, memory, and I/O resources.
- Applications remain isolated inside their own VM environments.

AASTU

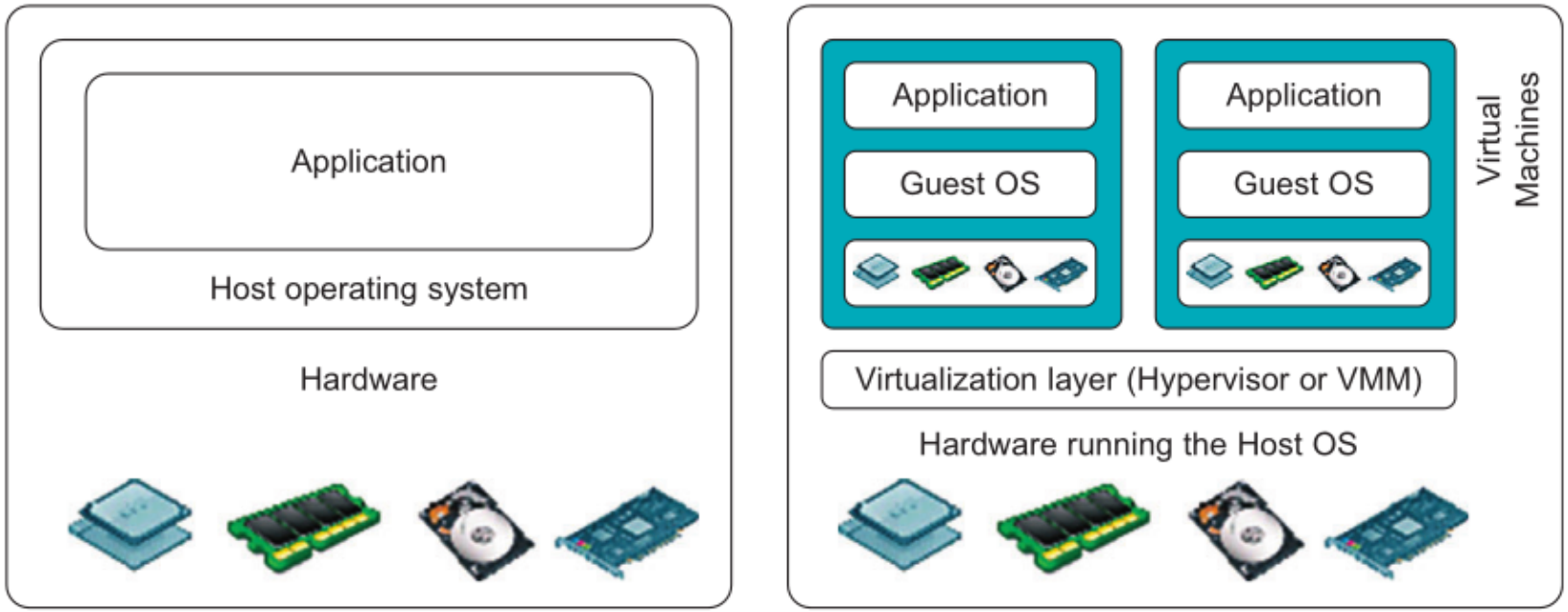
Levels of Virtualization Implementation

Where Can Virtualization Occur?

- Virtualization layer can be applied at different **operational levels**.
- Common levels include:
  - Instruction Set Architecture (ISA) Level
  - Hardware Level
  - Operating System Level
  - Library Support Level
  - Application Level
- Each level abstracts resources differently for VMs and applications.

AASTU

Implementation Levels of Virtualization



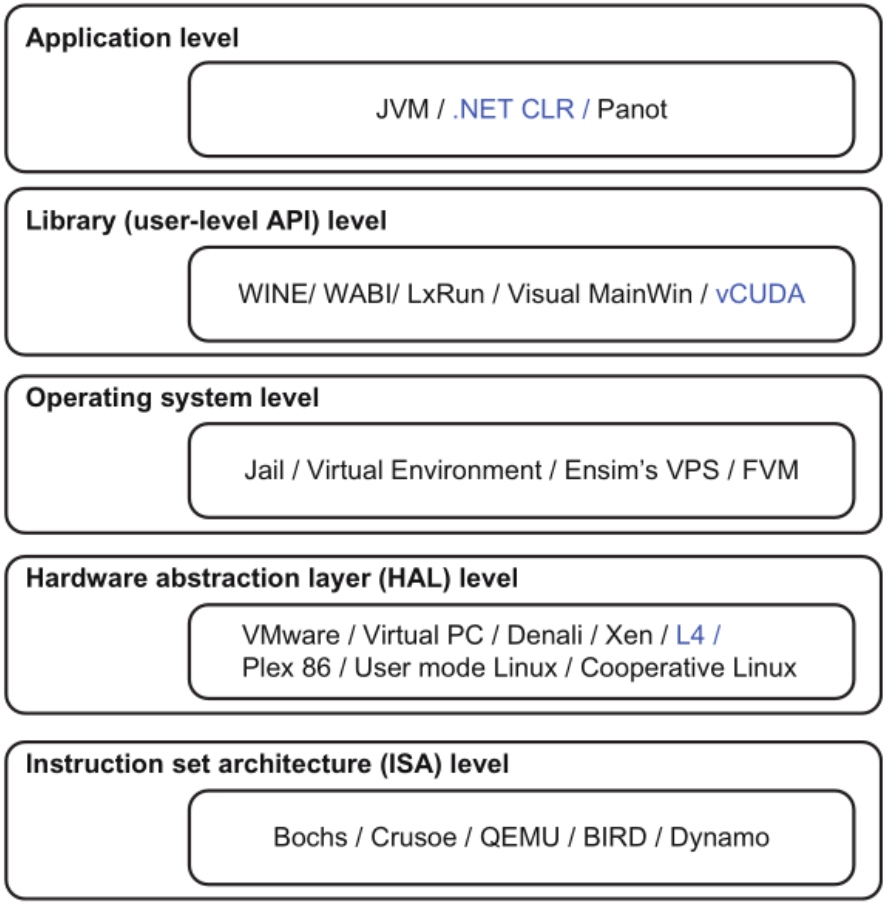
(a) Traditional computer

(b) After virtualization

K. Hwang, G. C. Fox, and J. J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2012.

AASTU

Implementation Levels of Virtualization



K. Hwang, G. C. Fox, and J. J. Dongarra,
Distributed and Cloud Computing:
From Parallel Processing
to the Internet of Things. Morgan Kaufmann, 2012.

AASTU

ISA Level Virtualization

Instruction Set Architecture (ISA) Level

- Emulates one ISA on top of another (e.g., MIPS code on x86 host).
- Supports running **legacy binaries** on new hardware.
- Implemented via:
  - Code interpretation (slow).
  - Dynamic binary translation (faster).
- Produces a **virtual ISA (V-ISA)** layer for portability.

AASTU

Hardware Level Virtualization

Directly on Hardware

- Virtualization occurs right above bare hardware.
- Provides VMs with **virtual CPUs, memory, and I/O**.
- Improves hardware utilization among multiple users.
- Early example: IBM VM/370 (1960s).
- Modern example: **Xen hypervisor** for x86 servers.

AASTU

OS Level Virtualization

Operating System Layer

- Provides an abstraction between OS and user apps.
- Creates **isolated containers** on a single server.
- Containers behave like independent servers.
- Widely used for:
  - Virtual hosting environments.
  - Server consolidation.

AASTU

Library Support Level Virtualization

Library Interface Layer

- Many applications use APIs via libraries instead of system calls.
- Virtualization intercepts API calls through **API hooks**.
- Examples:
  - WINE (Windows apps on UNIX).
  - vCUDA (GPU acceleration inside VMs).

AASTU

Application Level Virtualization

User-Application Layer

- Virtualizes applications as independent VMs.
- Often implemented via **High-Level Language (HLL) VMs**.
- Examples:
  - Java Virtual Machine (JVM).
  - Microsoft .NET CLR.
- Other approaches: application isolation, sandboxing, streaming.

AASTU

Relative Merits of Different Levels

Comparison Criteria

- Performance
- Application Flexibility
- Implementation Complexity
- Application Isolation

See Table 3.1 for detailed comparison.

AASTU

Virtual Machine Monitor (VMM)

What is a VMM?

- Hardware-level virtualization adds a layer between real hardware and OS.
- This layer is called the **Virtual Machine Monitor (VMM)**.
- Manages hardware resources of the system and intercepts program access to hardware.
- Enables multiple OS instances to run simultaneously on the same hardware.

AASTU

VMM Design Requirements

A VMM Must Provide:

1. An **identical environment** to the original machine.
2. Programs should run with only **minor performance decrease**.
3. **Complete control** of all system resources.

Note: Differences may arise from limited system resources or timing dependencies.

AASTU

Efficiency in VMMs

Key Point: Performance Matters

- Pure emulators/simulators are too slow for real machine use.
- **Efficient VMMs** execute the majority of instructions directly on hardware.
- Only exceptional cases require VMM intervention.
- Balance needed between **flexibility vs speed**.

AASTU

VMM Control of Resources

Responsibilities of a VMM

1. Allocate hardware resources (CPU, memory, I/O) to programs/VMs efficiently.
2. Prevent unauthorized access to resources not assigned to a VM.
3. Reclaim and redistribute resources when VMs terminate or underutilize them.
4. Monitor resource usage to ensure fair sharing among VMs.
5. Support performance isolation and security between VMs.

AASTU

Hypervisors and VMMs

Examples of Providers

- VMware Workstation, VMware ESX Server
- Xen Hypervisor (para-virtualization)
- KVM (Kernel-based Virtual Machine) item Microsoft Hyper-V (Windows-based hypervisor for enterprise)
- Oracle VirtualBox (desktop virtualization for testing/development)

K. Hwang, G. C. Fox, and J. J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2012.

AASTU

New Computing Mode: Cloud

- VM technology enables **cloud computing**.
- Cloud shifts hardware staffing costs to third parties.
- Challenges in cloud computing:
  - Variable CPU/VM requirements.
  - Slow instantiation of new VMs.

AASTU

Why OS-Level Virtualization?

Overcoming Hardware VM Limitations

- Hardware-level VMs are slow to initialize and store.
- Full hardware virtualization has:
  - Slow performance
  - Low density
  - Need for para-virtualization
- OS-level virtualization provides a faster, more resource-efficient solution.

AASTU

Virtual Execution Environments (VEs)

Containers / VPS / VE

- OS virtualization partitions physical resources for multiple isolated VMs.
- Each VE has:
  - Processes, file system, user accounts
  - Network interfaces, IPs, routing tables, firewall rules
- All VEs share the same OS kernel $\rightarrow$ called **single-OS image virtualization**.

AASTU

Advantages of OS-Level Virtualization

Benefits Compared to Hardware-Level VMs

- Minimal startup/shutdown costs and low resource requirements.
- High scalability: multiple VMs share one OS kernel.
- Host and VM states can be synchronized when needed.
- Faster initialization of VMs for cloud environments.

AASTU

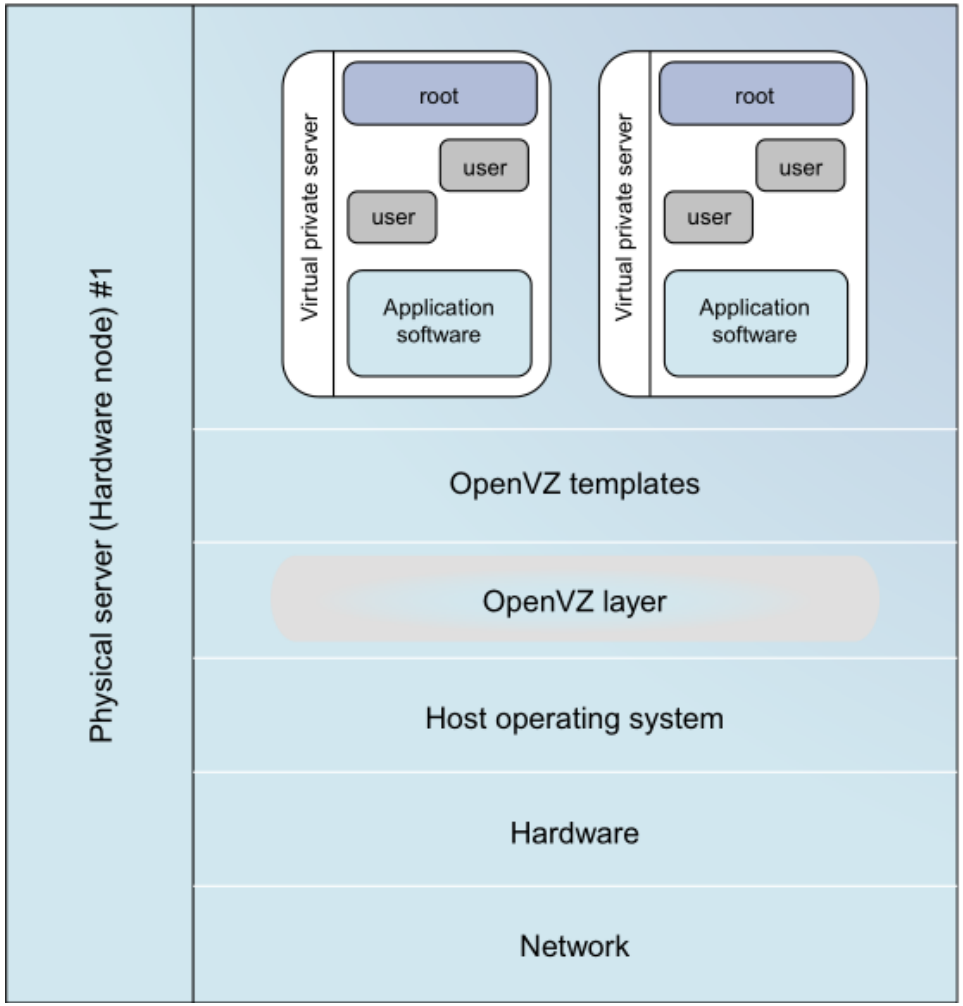
Disadvantages of OS-Level Virtualization

Limitations

- All VMs must belong to the same OS family.
- Different distributions allowed, but not different OS types (e.g., Windows on Linux not allowed).
- Resource duplication can incur significant overhead if not managed carefully.
- Often considered a secondary choice compared to hardware-assisted virtualization.

AASTU

The OpenVZ virtualization layer inside the host OS



K. Hwang, G. C. Fox, and J. J. Dongarra, *Distributed and Cloud Computing*, Morgan Kaufmann, 2012.

AASTU

What Are Containers?

Container Basics

- Containers bundle an application with parts of its OS environment.
- They run as a process on the host OS (e.g., Linux) rather than as a full OS.
- Each container sees itself as an independent machine:
  - Own IP address
  - Isolated filesystem
  - Isolated process tree

AASTU

Problems Containers Solve

Deployment Challenges

- **Configuration Management:** Test apps in the same environment they run in production.
- **Infrastructure Management:** Run multiple containers on a single machine.
- Containers allow scaling:
  - Move containers to larger servers when needed.
  - Run multiple containers on one physical host efficiently.

AASTU

Do Containers Use a Lot of Disk Space?

Minimal Footprint

- Containers typically include only the minimal OS required.
- Example: Alpine Linux ¡ 10 MB base image.
- Some applications can be fully packaged without an OS ("distroless" containers).
- Container runtimes manage disk usage across multiple containers efficiently.

AASTU

Docker and Containers

Container Management

- Docker is a popular tool for building, managing, and running containers.
- Docker Hub provides standardized container images.
- Containers are now largely standardized; Docker is one implementation.
- Alternatives like Podman are mostly CLI-compatible with Docker.

AASTU

History: From Emulators to VMs

Container Origins

- Early virtualization used **emulators** to simulate other computers.
- Emulation is slow but allowed running one computer inside another.
- Modern virtualization runs most instructions natively, only intercepting OS-level tasks.
- Enabled cloud computing: large physical machines can be partitioned into smaller virtual machines or containers.

AASTU

Docker Interactive Tutorial

Why Docker?

- Containers enable reliable, repeatable, and scalable deployments.
- They allow developers to manage infrastructure efficiently.
- Docker provides CLI tools to download, run, and build containers.
- Supports massive deployment with minimal effort.
- Real-world examples: AWS ECS, Kubernetes, Azure Container Instances.

AASTU

Registries, Repositories, and Tags

Docker Registry Concepts

- **Registry:** Stores container images online (e.g., Docker Hub, JFrog, AWS, Azure).
- **Repository:** Named location in a registry for a type of image.
- **Tag:** Identifies a specific version of an image (e.g., ubuntu:jammy).
- Fully qualified image name: `registry/repository:tag`.
- Tags help manage versions and updates reliably in production.

AASTU

Running Your First Container: Hello World

Command Example

- Run a simple container:

  `docker run johnnyb61820/hello-world`

- Pulls the image from Docker Hub and runs it.
- First run: downloads the image, saves locally, prints "Hello World!"
- Subsequent runs: uses local image, runs instantly.

AASTU

Understanding Containers on Your Machine

Key Commands

- List images: `docker image ls`
- List all containers: `docker container ls -a`
- Container = image + writable filesystem layer
- Each container runs as an isolated virtual machine
- Real-world example: Multiple containers running test scripts simultaneously without conflict.

AASTU

Removing Containers and Images

Cleanup Commands

- Auto-delete after run: `docker run --rm IMAGE`
- Delete non-running containers: `docker system prune`
- Add -a to remove unused images as well
- Containers share the read-only image, minimizing disk usage
- Real-world: Keep production servers clean and lightweight.

AASTU

Running a Docker Service: Web Server

Example: Simple HTTP Service

- Run a container with port mapping:

  `docker run -p 8080:8070 johnnyb61820/simple-web-server`
- Access via browser: `http://localhost:8080/`
- Run in background with `-d` flag
- Stop container: `docker container stop CONTAINER/D`

AASTU

Resource Isolation in Containers

Key Concept

- Containers provide strong isolation for resources, ensuring that each application runs independently.
- Each container can have dedicated CPU, memory, disk, and network allocation.

Real-World Example

- Containers isolated with resource limits; Kubernetes enforces this via namespaces and cgroups.

AASTU

Resource Isolation Benefits

Why It Matters

- Ensures containers do not interfere with each other.
- Improves security, stability, and predictability
- Prevents resource exhaustion by a single container
- Simplifies debugging and monitoring

AASTU

Real-World Examples of Resource Isolation

Examples in Practice

- Multiple web servers running on a single host, each with dedicated CPU and memory.
- Database containers isolated from web app containers to prevent performance interference.
- Kubernetes enforces resource quotas and limits for pods in production environments.
- CI/CD pipelines and cloud multi-tenant environments run in isolated containers.

AASTU

Cloud Infrastructure Overview

Key Concepts

- Provides on-demand computing resources over the Internet.
- Resources include virtual machines, containers, storage, and networking.
- Supports scalable, elastic, pay-as-you-go deployment.
- Enables rapid deployment of applications and services.
- Supports container orchestration and microservices architectures.
- Improves operational efficiency and reduces infrastructure costs.

AASTU

Cloud Infrastructure Examples

Popular Cloud Providers

- AWS: EC2 (VMs), ECS Fargate (containers), Lambda (serverless)
- Google Cloud: Compute Engine, GKE (Kubernetes), Cloud Run
- Azure: Virtual Machines, Container Instances, Azure Functions

Real-World Use Cases

- Netflix, Spotify, Airbnb run containerized apps on cloud.
- Startups use cloud for scalable multi-tenant applications.

AASTU

Multi-Tier Cloud Application

Application Tiers

- Web tier: Containerized web app in AWS ECS or Google Cloud Run
- Database tier: Managed RDS (AWS) or Cloud SQL (GCP)
- Cache tier: Redis or Memcached container in AWS Fargate or GKE

Supporting Components

- Load Balancer: AWS ALB or GCP Load Balancer routes traffic
- Auto-Scaling and Monitoring: Dynamically adjust containers and track performance (CloudWatch, Stackdriver)

AASTU

Questions?

Thank you for your attention!

AASTU

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.

AASTU