 Light

Create Release passing | Translate README failing | Generate HTML and PDF passing
Deploy Webpage passing | Generate GitBook Docs passing | PSScriptAnalyzer passing
Slack Notification passing

LICENSE MIT | FORKS 2 | STARS 5 | CONTRIBUTORS 2 | ISSUES 0 OPEN
LINKEDIN

LEARNING LPIC-3 305-300

Explore the docs »

Web Site - Code Page - Gitbook - Report Bug - Request Feature

Summary

▶ **TABLE OF CONTENT**

About Project

This project aims to help students or professionals to learn the main concepts of GNULin
and free software
Some GNULinux distributions like Debian and RPM will be covered
Installation and configuration of some packages will also be covered
By doing this you can give the whole community a chance to benefit from your changes.
Access to the source code is a precondition for this.
Use vagrant for up machines and execute labs and practice content in this article.
I have published in folder Vagrant a Vagrantfile with what is necessary
for you to upload an environment for studies

(back to top)

Getting Started

For starting the learning, see the documentation above.

Prerequisites

- Git
- VMware Workstation
- Vagrant VMWare Utility
- Vagrant

Installation

Clone the repo

```
git clone https://github.com/marcossilvestrini/learning-lpic-3-305-300.git
cd learning-lpic-3-305-300
```

Customize a template *Vagrantfile-topic-XXX*. This file contains a vms configuration for labs. Example:

- File Vagrantfile-topic-351
  - vm.clone_directory = "<your_driver_letter>:\ \<to_machine>\#{VM_NAME}-instance-1"
    Example: vm.clone_directory = "E:\Servers\VMWare\#{VM_NAME}-instance-1"
  - vm.vmx["memsize"] = ""
  - vm.vmx["numvcpus"] = ""
  - vm.vmx["cpuid.coresPerSocket"] = ""

Customize network configuration in files configs/network.

Usage

Use this repository for get learning about LPIC-3 305-300 exam

For up and down

Switch a *Vagrantfile-topic-xxx* template and copy for a new file with name *Vagrantfile*

```
cd vagrant && vagrant up
cd vagrant && vagrant destroy -f
```

For reboot vms

```
cd vagrant && vagrant reload
```

Important: *If you reboot vms without vagrant, shared folder not mount after boot.*

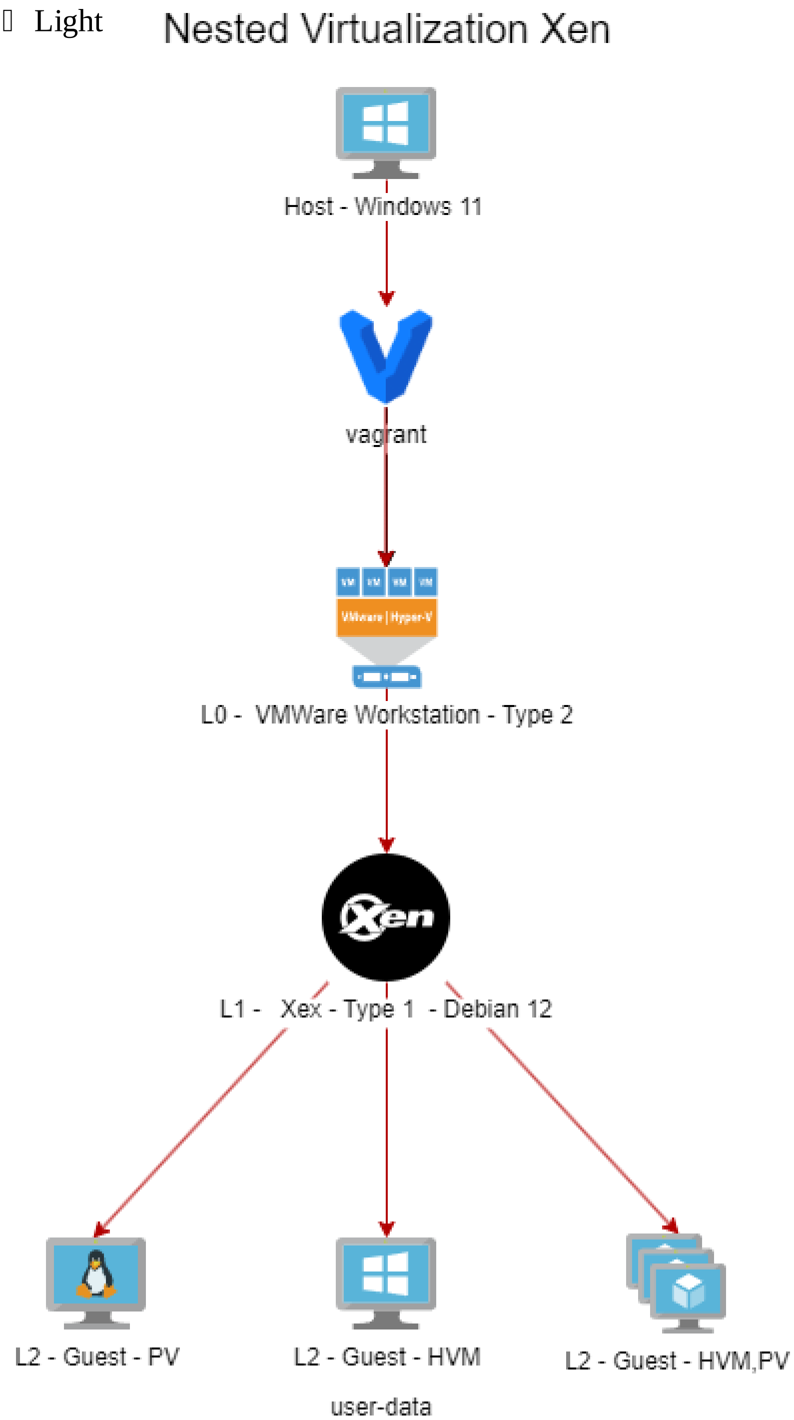
Use powershell for up and down

If you use Windows platform, I create a powershell script for up and down vms.

```
vagrant/up.ps1
vagrant/destroy.ps1
```

Infrastructure Schema Topic 351

Light



(back to top)

Roadmap

- [x] Create repository
- [x] Create scripts for provisioning labs
- [x] Create examples about Topic 351
- [ ] Create examples about Topic 352
- [ ] Create examples about Topic 353
- [ ] Upload simulated itexam

 Light

Four Essential Freedoms

0.The freedom to run the program as you wish, for any purpose (freedom 0).
1.The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1).
Access to the source code is a precondition for this.
2.The freedom to redistribute copies so you can help others (freedom 2).
3.freedom to distribute copies of your modified versions to others (freedom 3).

Inspect commands

```
type COMMAND
apropos COMMAND
whatis COMMAND --long
whereis COMMAND
COMMAND --help, --h
man COMMAND
```

(back to top)

Topic 351: Full Virtualization

The difference between type-1 and type-2 hypervisors

351.1 Virtualization Concepts and Theory

Weight: 6

Description: Candidates should know and understand the general concepts, theory and terminology of virtualization. This includes Xen, QEMU and libvirt terminology.

Key Knowledge Areas:

- Understand virtualization terminology
- Understand the pros and cons of virtualization
- Understand the various variations of Hypervisors and Virtual Machine Monitors
- Understand the major aspects of migrating physical to virtual machines
- Understand the major aspects of migrating virtual machines between host systems
- Understand the features and implications of virtualization for a virtual machine, such as snapshotting, pausing, cloning and resource limits
- Awareness of oVirt, Proxmox, systemd-machined and VirtualBox
- Awareness of Open vSwitch

351.1 Cited Objects

```
Hypervisor
Hardware Virtual Machine (HVM)
Paravirtualization (PV)
Emulation and Simulation
CPU flags
/proc/cpuinfo
Migration (P2V, V2V)
```

Hypervisors

Type 1 Hypervisor (Bare-Metal Hypervisor)

Type 1 Definition

Runs directly on the host's physical hardware, providing a base layer to manage VMs without the need for a host operating system.

Type 1 Characteristics

- High performance and efficiency.
- Lower latency and overhead.
- Often used in enterprise environments and data centers.

Type 1 Examples

- VMware ESXi: A robust and widely used hypervisor in enterprise settings.
- Microsoft Hyper-V: Integrated with Windows Server, offering strong performance and management features.

- Xen: An open-source hypervisor used by many cloud service providers.
- KVM (Kernel-based Virtual Machine): Integrated into the Linux kernel, providing high performance for Linux-based systems.

Type 2 Hypervisor (Hosted Hypervisor)

Type 2 Definition

Runs on top of a conventional operating system, relying on the host OS for resource management and device support.

Type 2 Characteristics

- Easier to set up and use, especially on personal computers.
- More flexible for development, testing, and smaller-scale deployments.
- Typically less efficient than Type 1 hypervisors due to additional overhead from the host OS.

Type 2 Examples

- VMware Workstation: A powerful hypervisor for running multiple operating systems on a single desktop.
- Oracle VirtualBox: An open-source hypervisor known for its flexibility and ease of use.
- Parallels Desktop: Designed for Mac users to run Windows and other operating systems alongside macOS.
- QEMU (Quick EMUlator): An open-source emulator and virtualizer, often used in conjunction with KVM.

Key Differences Between Type 1 and Type 2 Hypervisors

- Deployment Environment:
  - Type 1 hypervisors are commonly deployed in data centers and enterprise environments due to their direct interaction with hardware and high performance.
  - Type 2 hypervisors are more suitable for personal use, development, testing, and small-scale virtualization tasks.
- Performance:
  - Type 1 hypervisors generally offer better performance and lower latency because they do not rely on a host OS.
  - Type 2 hypervisors may experience some performance degradation due to the overhead of running on top of a host OS.
- Management and Ease of Use:
  - Type 1 hypervisors require more complex setup and management but provide advanced features and scalability for large-scale deployments.

Light

- Type 2 hypervisors are easier to install and use, making them ideal for individual u
  and smaller projects.

Migration Types

In the context of hypervisors, which are technologies used to create and manage virtual machines, the terms P2V migration and V2V migration are common in virtualization environments.
They refer to processes of migrating systems between different types of platforms.

P2V - Physical to Virtual Migration

P2V migration refers to the process of migrating a physical server to a virtual machine.
In other words, an operating system and its applications, running on dedicated physical hardware, are "converted" and moved to a virtual machine that runs on a hypervisor (such as VMware, Hyper-V, KVM, etc.).

- Example: You have a physical server running a Windows or Linux system, and you want to move it to a virtual environment, like a cloud infrastructure or an internal virtualization server. The process involves copying the entire system state, including the operating system, drivers, and data, to create an equivalent virtual machine that can run as if it were on the physical hardware.

V2V - Virtual to Virtual Migration

V2V migration refers to the process of migrating a virtual machine from one hypervisor to another.
In this case, you already have a virtual machine running in a virtualized environment (like VMware), and you want to move it to another virtualized environment (for example, to Hyper-V or to a new VMware server).

- Example: You have a virtual machine running on a VMware virtualization server, but you decide to migrate it to a Hyper-V platform. In this case, the V2V migration converts the virtual machine from one format or hypervisor to another, ensuring it can continue running correctly.

HVM and Paravirtualization

Hardware-assisted Virtualization (HVM)

HVM Definition

HVM leverages hardware extensions provided by modern CPUs to virtualize hardware, enabling the creation and management of VMs with minimal performance overhead.

HVM Key Characteristics

- **Hardware Support**: Requires CPU support for virtualization extensions such as Intel VT-x or AMD-V.

- **Full Virtualization:** VMs can run unmodified guest operating systems, as the hypervisor provides a complete emulation of the hardware environment.
- **Performance:** Typically offers near-native performance because of direct execution of guest code on the CPU.
- **Isolation:** Provides strong isolation between VMs since each VM operates as if it has its own dedicated hardware.

HVM Examples

VMware ESXi, Microsoft Hyper-V, KVM (Kernel-based Virtual Machine).

HVM Advantages

- **Compatibility:** Can run any operating system without modification.
- **Performance:** High performance due to hardware support.
- **Security:** Enhanced isolation and security features provided by hardware.

HVM Disadvantages

- **Hardware Dependency:** Requires specific hardware features, limiting compatibility with older systems.
- **Complexity:** May involve more complex configuration and management.

Paravirtualization

Paravirtualization Definition

Paravirtualization involves modifying the guest operating system to be aware of the virtual environment, allowing it to interact more efficiently with the hypervisor.

Paravirtualization Key Characteristics

- **Guest Modification:** Requires changes to the guest operating system to communicate directly with the hypervisor using hypercalls.
- **Performance:** Can be more efficient than traditional full virtualization because it reduces the overhead associated with emulating hardware.
- **Compatibility:** Limited to operating systems that have been modified for paravirtualization.

Paravirtualization Examples

Xen with paravirtualized guests, VMware tools in certain configurations, and some KVM configurations.

Paravirtualization Advantages

- **Efficiency:** Reduces the overhead of virtualizing hardware, potentially offering better performance for certain workloads.

- **Resource Utilization:** More efficient use of system resources due to direct communi between the guest OS and hypervisor.

Paravirtualization Disadvantages

- **Guest OS Modification:** Requires modifications to the guest OS, limiting compatibility to supported operating systems.
- **Complexity:** Requires additional complexity in the guest OS for hypercall implementations.

Key Differences

Guest OS Requirements

- **HVM:** Can run unmodified guest operating systems.
- **Paravirtualization:** Requires guest operating systems to be modified to work with the hypervisor.

Performance

- **HVM:** Typically provides near-native performance due to hardware-assisted execution.
- **Paravirtualization:** Can offer efficient performance by reducing the overhead of hardware emulation, but relies on modified guest OS.

Hardware Dependency

- **HVM:** Requires specific CPU features (Intel VT-x, AMD-V).
- **Paravirtualization:** Does not require specific CPU features but needs modified guest OS.

Isolation

- **HVM:** Provides strong isolation using hardware features.
- **Paravirtualization:** Relies on software-based isolation, which may not be as robust as hardware-based isolation.

Complexity

- **HVM:** Generally more straightforward to deploy since it supports unmodified OS.
- **Paravirtualization:** Requires additional setup and modifications to the guest OS, increasing complexity.

NUMA (Non-Uniform Memory Access)

NUMA (Non-Uniform Memory Access) is a memory architecture used in multiprocessor systems to optimize memory access by processors.
In a NUMA system, memory is distributed unevenly among processors, meaning that each

processor has faster access to a portion of memory (its "local memory") than to memory tha physically further away (referred to as "remote memory") and associated with other processors.

Key Features of NUMA Architecture

1. **Local and Remote Memory**: Each processor has its own local memory, which it can access more quickly. However, it can also access the memory of other processors, although this takes longer.
2. **Differentiated Latency**: The latency of memory access varies depending on whether the processor is accessing its local memory or the memory of another node. Local memory access is faster, while accessing another node’s memory (remote) is slower.
3. **Scalability**: NUMA architecture is designed to improve scalability in systems with many processors. As more processors are added, memory is also distributed, avoiding the bottleneck that would occur in a uniform memory access (UMA) architecture.

Advantages of NUMA

- Better Performance in Large Systems: Since each processor has local memory, it can work more efficiently without competing as much with other processors for memory access.
- Scalability: NUMA allows systems with many processors and large amounts of memory to scale more effectively compared to a UMA architecture.

Disadvantages

- Programming Complexity: Programmers need to be aware of which regions of memory are local or remote, optimizing the use of local memory to achieve better performance.
- Potential Performance Penalties: If a processor frequently accesses remote memory, performance may suffer due to higher latency. This architecture is common in high-performance multiprocessor systems, such as servers and supercomputers, where scalability and memory optimization are critical.

Opensource Solutions

- oVirt: https://www.ovirt.org/
- Proxmox: https://www.proxmox.com/en/proxmox-virtual-environment/overview
- Oracle VirtualBox: https://www.virtualbox.org/
- Open vSwitch: https://www.openvswitch.org/

Types of Virtualization

Hardware Virtualization (Server Virtualization)

HV Definition

Abstracts physical hardware to create virtual machines (VMs) that run separate operating sy and applications.

HV Use Cases

Data centers, cloud computing, server consolidation.

HV Examples

VMware ESXi, Microsoft Hyper-V, KVM.

Operating System Virtualization (Containerization)

Containerization Definition

Allows multiple isolated user-space instances (containers) to run on a single OS kernel.

Containerization Use Cases

Microservices architecture, development and testing environments.

Containerization Examples

Docker, Kubernetes, LXC.

Network Virtualization

Network Virtualization Definition

Combines hardware and software network resources into a single, software-based administrative entity.

Network Virtualization Use Cases

Software-defined networking (SDN), network function virtualization (NFV).

Network Virtualization Examples

VMware NSX, Cisco ACI, OpenStack Neutron.

Storage Virtualization

Storage VirtualizationDefinition

Pools physical storage from multiple devices into a single virtual storage unit that can be managed centrally.

Storage VirtualizationDefinition Use Cases

Data management, storage optimization, disaster recovery.

Storage VirtualizationDefinition Examples

IBM SAN Volume Controller, VMware vSAN, NetApp ONTAP.

Desktop Virtualization

Desktop Virtualization Definition

Allows a desktop operating system to run on a virtual machine hosted on a server.

Desktop Virtualization Definition Use Cases

Virtual desktop infrastructure (VDI), remote work solutions.

Desktop Virtualization Definition Examples

Citrix Virtual Apps and Desktops, VMware Horizon, Microsoft Remote Desktop Services.

Application Virtualization

Application VirtualizationDefinition

Separates applications from the underlying hardware and operating system, allowing them to run in isolated environments.

Application VirtualizationDefinition Use Cases

Simplified application deployment, compatibility testing.

Application VirtualizationDefinition Examples

VMware ThinApp, Microsoft App-V, Citrix XenApp.

Data Virtualization

Data VirtualizationDefinition

Integrates data from various sources without physically consolidating it, providing a unified view for analysis and reporting.

Data VirtualizationDefinition Use Cases

Business intelligence, real-time data integration.

Data VirtualizationDefinition Examples

Denodo, Red Hat JBoss Data Virtualization, IBM InfoSphere.

Benefits of Virtualization

- Resource Efficiency: Better utilization of physical resources.

 Light

- Cost Savings: Reduced hardware and operational costs.
- Scalability: Easy to scale up or down according to demand.
- Flexibility: Supports a variety of workloads and applications.
- Disaster Recovery: Simplified backup and recovery processes.
- Isolation: Improved security through isolation of environments.

Emulation

Emulation involves simulating the behavior of hardware or software on a different platform than originally intended.

This process allows software designed for one system to run on another system that may have different architecture or operating environment.

While emulation provides versatility by enabling the execution of unmodified guest operating systems or applications, it often comes with performance overhead.

This overhead arises because the emulated system needs to interpret and translate instructions meant for the original system into instructions compatible with the host system. As a result, emulation can be slower than native execution, making it less efficient for resource-intensive tasks.

Despite this drawback, emulation remains valuable for running legacy software, testing applications across different platforms, and facilitating cross-platform development.

systemd-machined

The systemd-machined service is dedicated to managing virtual machines and containers within the systemd ecosystem. It provides essential functionalities for controlling, monitoring, and maintaining virtual instances, offering robust integration and efficiency within Linux environments.

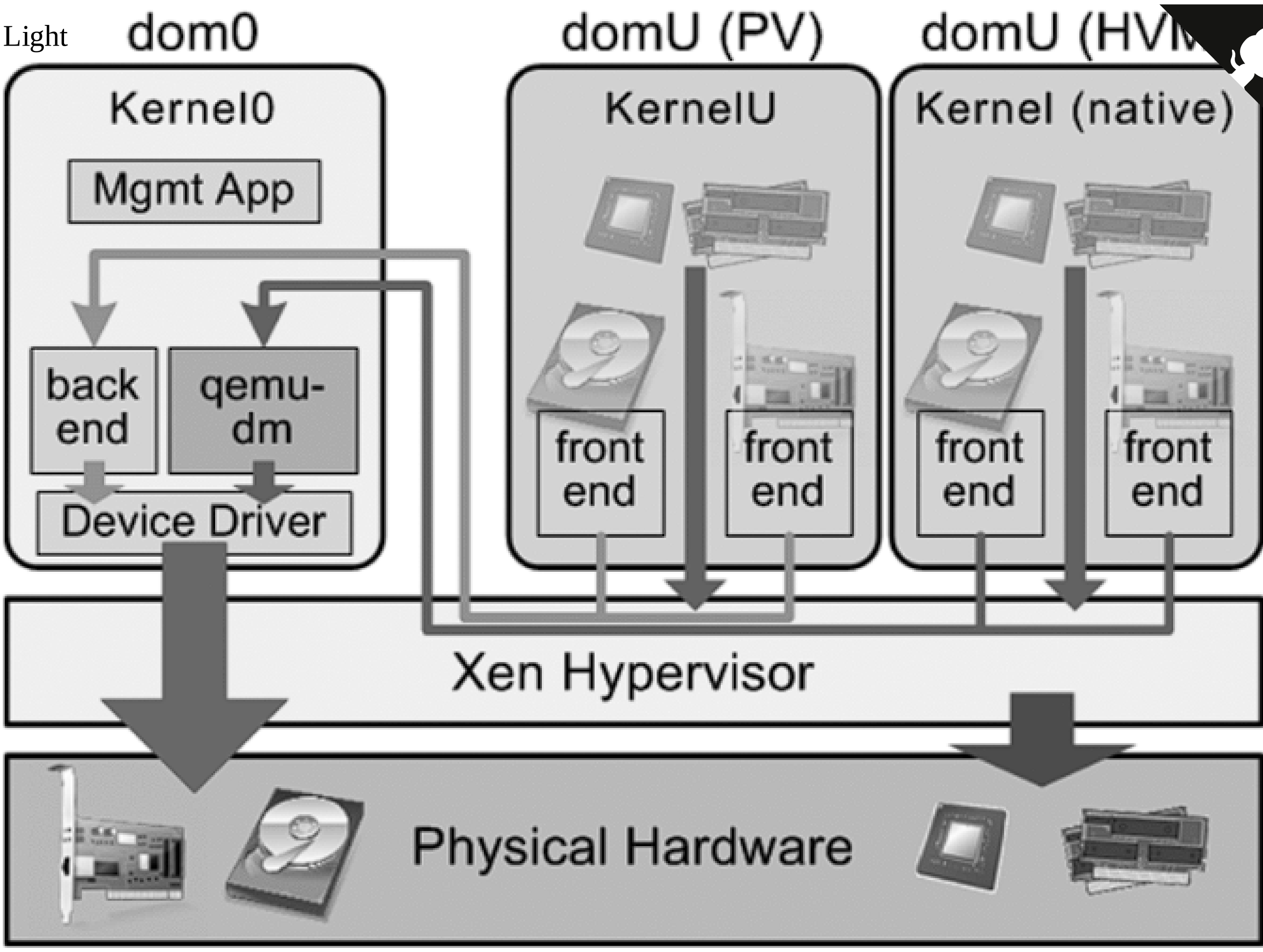
(back to sub Topic 351.1)

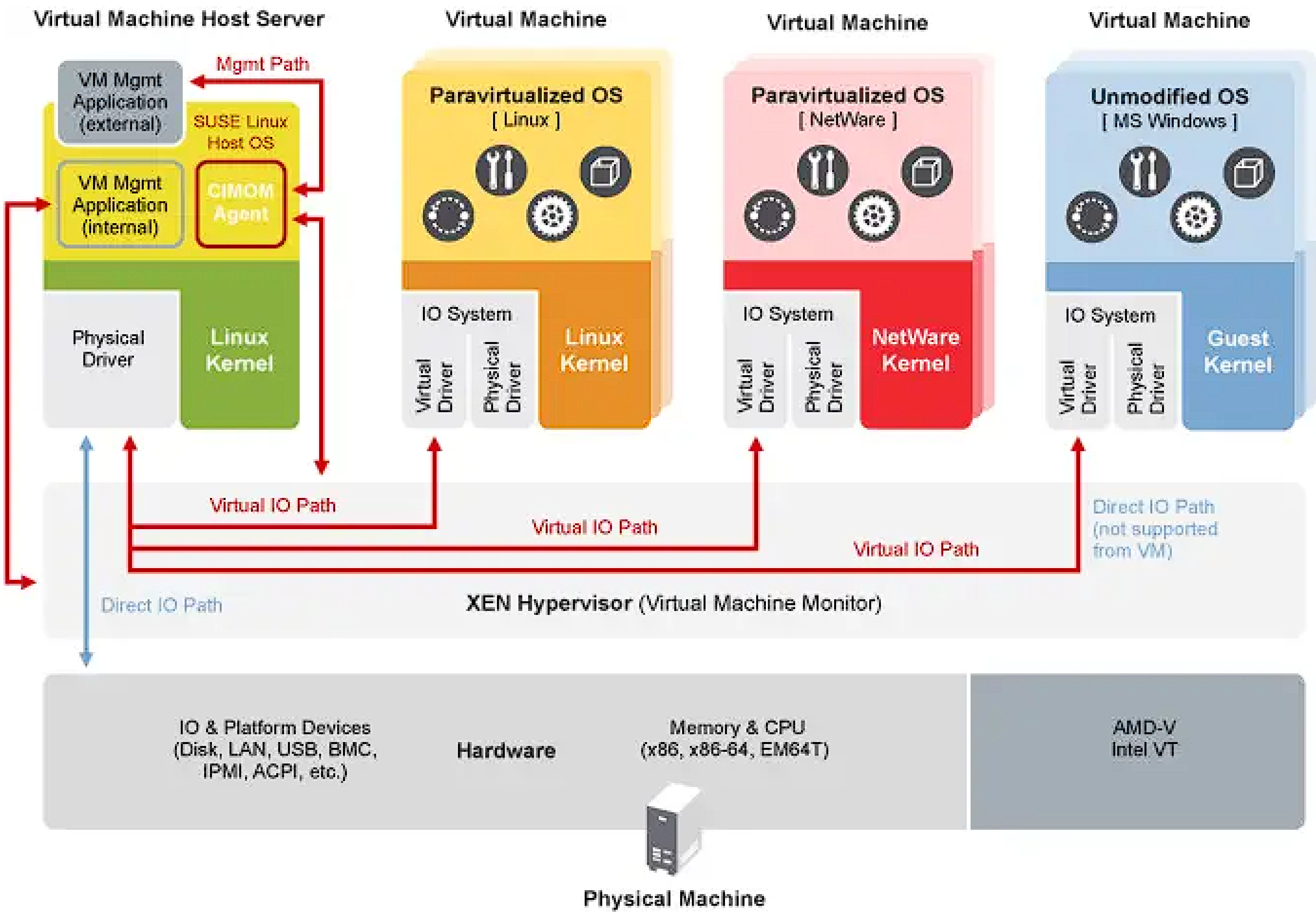
(back to Topic 351)

(back to top)

351.2 Xen

 Light


dom0
domU (PV)
domU (HVM
Kernel0
Mgmt App
back end
qemu-dm
Device Driver
KernelU
front end
front end
Kernel (native)
front end
front end
Xen Hypervisor
Physical Hardware


Virtual Machine Host Server
Virtual Machine
Virtual Machine
Virtual Machine
VM Mgmt Application (external)
Mgmt Path
SUSE Linux Host OS
VM Mgmt Application (internal)
CIMOM Agent
Physical Driver
Linux Kernel
Paravirtualized OS [Linux]
IO System
Virtual Driver
Physical Driver
Linux Kernel
Paravirtualized OS [NetWare]
IO System
Virtual Driver
Physical Driver
NetWare Kernel
Unmodified OS [MS Windows]
IO System
Virtual Driver
Physical Driver
Guest Kernel
Virtual IO Path
Virtual IO Path
Virtual IO Path
Direct IO Path (not supported from VM)
Direct IO Path
XEN Hypervisor (Virtual Machine Monitor)
IO & Platform Devices (Disk, LAN, USB, BMC, IPMI, ACPI, etc.)
Hardware
Memory & CPU (x86, x86-64, EM64T)
AMD-V Intel VT
Physical Machine

Weight: 3

Description: Candidates should be able to install, configure, maintain, migrate and troubleshoot Xen installations. The focus is on Xen version 4.x.

Key Knowledge Areas:

- Understand architecture of Xen, including networking and storage
- Basic configuration of Xen nodes and domains
- Basic management of Xen nodes and domains
- Basic troubleshooting of Xen installations
- Awareness of XAPI
- Awareness of XenStore
- Awareness of Xen Boot Parameters
- Awareness of the xm utility

Xen

Xen is an open-source type-1 (bare-metal) hypervisor, which allows multiple operating systems to run concurrently on the same physical hardware.
Xen provides a layer between the physical hardware and virtual machines (VMs), enabling efficient resource sharing and isolation.

- **Architecture:** Xen operates with a two-tier system where Domain 0 (Dom0) is the privileged domain with direct hardware access and manages the hypervisor. Other virtual machines, called Domain U (DomU), run guest operating systems and are managed by Dom0.
- **Types of Virtualization:** Xen supports both paravirtualization (PV), which requires modified guest OS, and hardware-assisted virtualization (HVM), which uses hardware extensions (e.g., Intel VT-x or AMD-V) to run unmodified guest operating systems. Xen is widely used in cloud environments, notably by Amazon Web Services (AWS) and other large-scale cloud providers.

XenSource

XenSource was the company founded by the original developers of the Xen hypervisor at the University of Cambridge to commercialize Xen.
The company provided enterprise solutions based on Xen and offered additional tools and support to enhance Xen's capabilities for enterprise use.

- **Acquisition by Citrix**: In 2007, XenSource was acquired by Citrix Systems, Inc. Citrix used Xen technology as the foundation for its Citrix XenServer product, which became a popular enterprise-grade virtualization platform based on Xen.
- **Transition**: After the acquisition, the Xen project continued as an open-source project, while Citrix focused on commercial offerings like XenServer, leveraging XenSource technology.

Xen Project

Xen Project refers to the open-source community and initiative responsible for developing and maintaining the Xen hypervisor after its commercialization.
The Xen Project operates under the Linux Foundation, with a focus on building, improving, and supporting Xen as a collaborative, community-driven effort.

- **Goals:** The Xen Project aims to advance the hypervisor by improving its performance, security, and feature set for a wide range of use cases, including cloud computing, security-focused virtualization (e.g., Qubes OS), and embedded systems.
- **Contributors:** The project includes contributors from various organizations, including major cloud providers, hardware vendors, and independent developers.
- **XAPI and XenTools:** The Xen Project also includes tools such as XAPI (XenAPI), which is used for managing Xen hypervisor installations, and various other utilities for system management and optimization.

XenStore

Xen Store is a critical component of the Xen Hypervisor.
Essentially, Xen Store is a distributed key-value database used for communication and information sharing between the Xen hypervisor and the virtual machines (also known as domains) it manages.

Here are some key aspects of Xen Store:

- **Inter-Domain Communication:** Xen Store enables communication between domains, such as Dom0 (the privileged domain that controls hardware resources) and DomUs (user domains, which are the VMs). This is done through key-value entries, where each domain can read or write information.

- **Configuration Management:** It is used to store and access configuration information, such as virtual devices, networking, and boot parameters. This facilitates the dynamic management and configuration of VMs.

- **Events and Notifications:** Xen Store also supports event notifications. When a particular key or value in the Xen Store is modified, interested domains can be notified to react to the changes. This is useful for monitoring and managing resources.

- Simple API: Xen Store provides a simple API for reading and writing data, making it easy for developers to integrate their applications with the Xen virtualization system.

XAPI

XAPI, or XenAPI, is the application programming interface (API) used to manage the Xen Hypervisor and its virtual machines (VMs).
XAPI is a key component of XenServer (now known as Citrix Hypervisor) and provides a standardized way to interact with the Xen hypervisor to perform operations such as creating, configuring, monitoring, and controlling VMs.

Here are some important aspects of XAPI:

- **VM Management:** XAPI allows administrators to programmatically create, delete, start, and stop virtual machines.

- **Automation:** With XAPI, it's possible to automate the management of virtual resources, including networking, storage, and computing, which is crucial for large cloud environments.

- **Integration:** XAPI can be integrated with other tools and scripts to provide more efficient and customized administration of the Xen environment.

- **Access Control:** XAPI also provides access control mechanisms to ensure that only authorized users can perform specific operations in the virtual environment.

XAPI is the interface that enables control and automation of the Xen Hypervisor, making it easier to manage virtualized environments.

Xen Summary

- **Xen:** The core hypervisor technology enabling virtual machines to run on physical hardware.
- **XenSource:** The company that commercialized Xen, later acquired by Citrix, leading to the development of Citrix XenServer.
- **Xen Project:** The open-source initiative and community that continues to develop and maintain the Xen hypervisor under the Linux Foundation.
- **XenStore:** Xen Store acts as a communication and configuration intermediary between the Xen hypervisor and the VMs, streamlining the operation and management of virtualized environments.
- **XAPI** is the interface that enables control and automation of the Xen Hypervisor, making it easier to manage virtualized environments.

Domain0 (Dom0)

Domain0, or Dom0, is the control domain in a Xen architecture. It manages other domains (DomUs) and has direct access to hardware.
Dom0 runs device drivers, allowing DomUs, which lack direct hardware access, to communicate with devices. Typically, it is a full instance of an operating system, like Linux, and is essential for Xen hypervisor operation.

DomainU (DomU)

DomUs are non-privileged domains that run virtual machines.
They are managed by Dom0 and do not have direct access to hardware. DomUs can be configured to run different operating systems and are used for various purposes, such as application servers and development environments. They rely on Dom0 for hardware interaction.

PV-DomU (Paravirtualized DomainU)

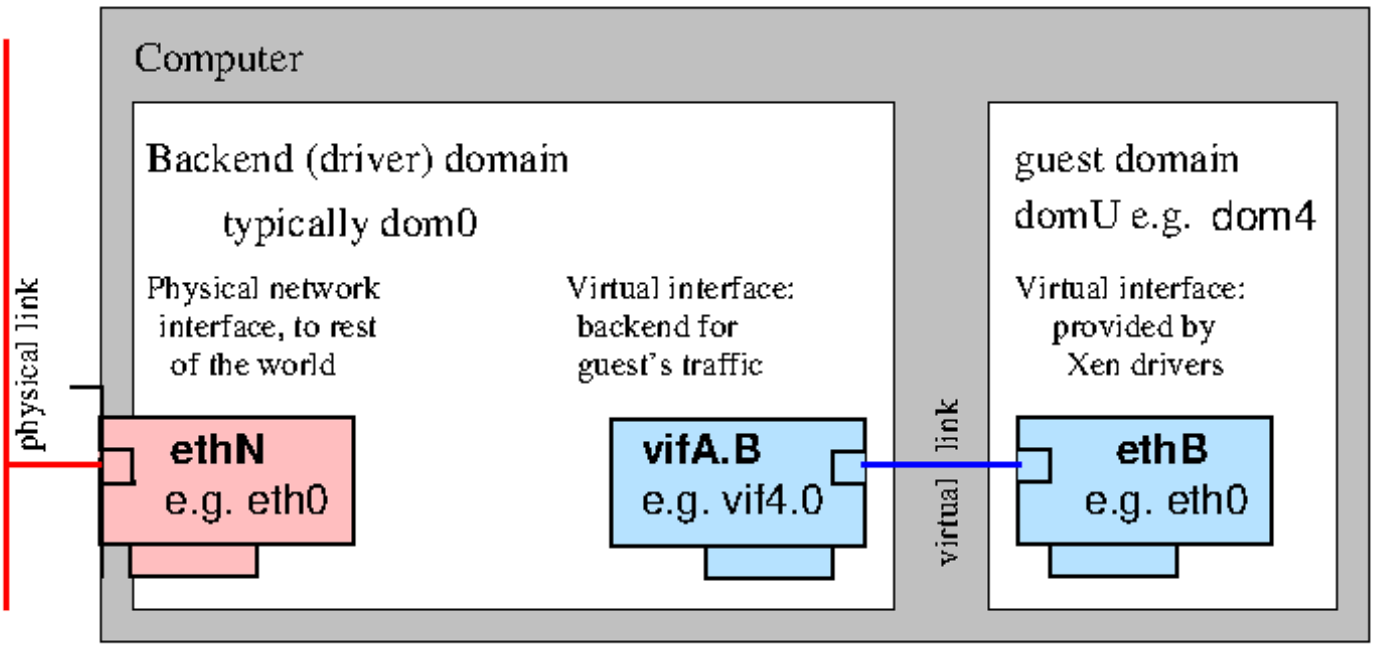
PV-DomUs use a technique called paravirtualization. In this model, the DomU operating system is modified to be aware that it runs in a virtualized environment, allowing it to communicate directly with the hypervisor for optimized performance.
This results in lower overhead and better efficiency compared to full virtualization.

HVM-DomU (Hardware Virtual Machine DomainU)

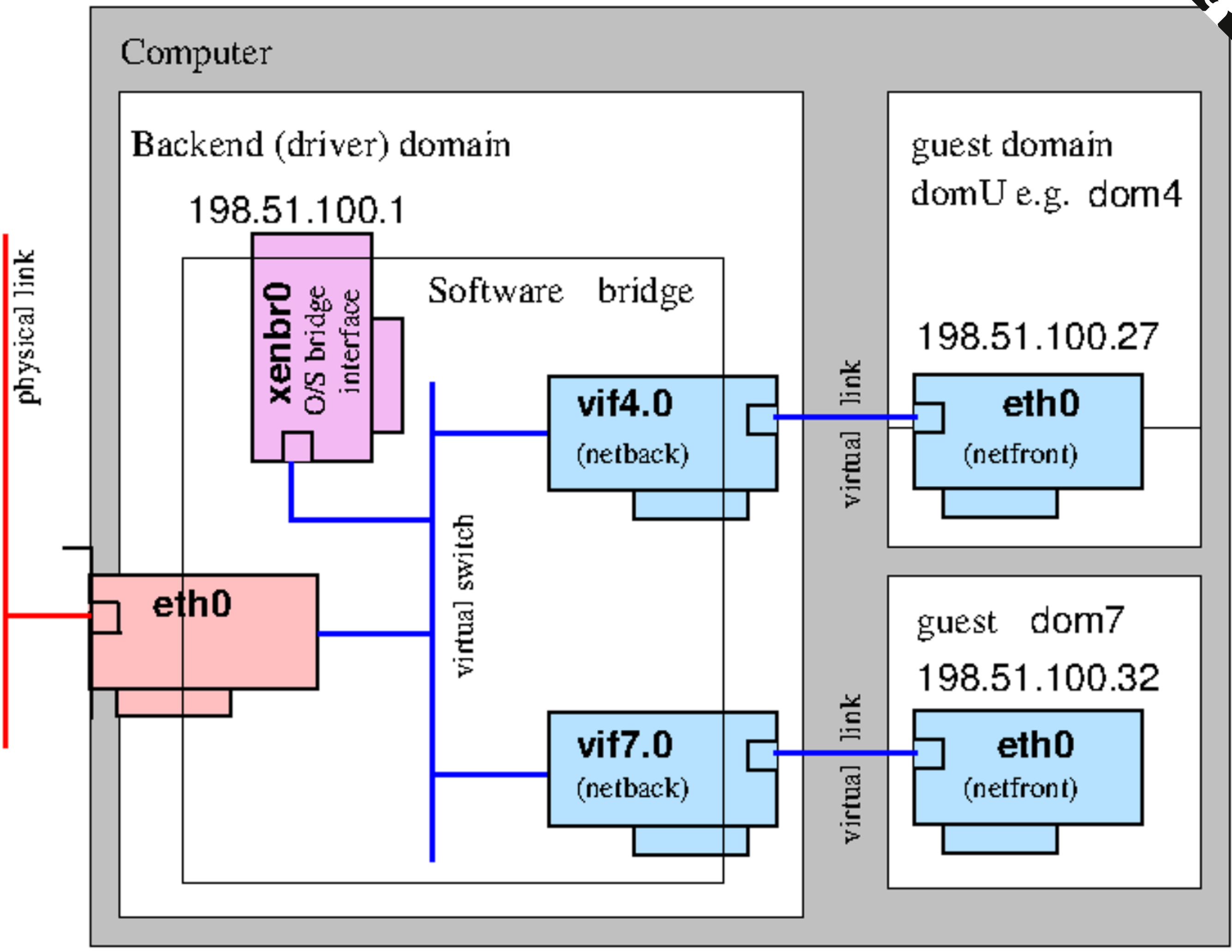
HVM-DomUs are virtual machines that utilize full virtualization, allowing unmodified operating systems to run. The Xen hypervisor provides hardware emulation for these DomUs, enabling them to run any operating system that supports the underlying hardware architecture.
While this offers greater flexibility, it can result in higher overhead compared to PV-DomUs.

Xen Network

Paravirtualised Network Devices



Bridging



351.2 Cited Objects

```
Domain0 (Dom0), DomainU (DomU)
PV-DomU, HVM-DomU
/etc/xen/
xl
xl.cfg
xl.conf # Xen global configurations
xentop
oxenstored # Xenstore configurations
```

351.2 Notes

```
# Xen Settings
/etc/xen/
/etc/xen/xl.conf - Main general configuration file for Xen
/etc/xen/oxenstored.conf - Xenstore configurations

# VM Configurations
/etc/xen/xlexample.pvlinux
/etc/xen/xlexample.hvm
```

```
# Service Configurations
/etc/default/xen
/etc/default/xendomains

# xen-tools configurations
/etc/xen-tools/
/usr/share/xen-tools/

# docs
xl(1)
xl.conf(5)
xlcpupool.cfg(5)
xl-disk-configuration(5)
xl-network-configuration(5)
xen-tscmode(7)

# initialized domains auto
/etc/default/xendomains
   XENDOMAINS_AUTO=/etc/xen/auto

/etc/xen/auto/


# set domain for up after xen reboot
## create folder auto
cd /etc/xen && mkdir -p auto && cd auto

# create simbolic link
ln -s /etc/xen/lpic3-pv-guest /etc/xen/auto/lpic3-pv-guest
```

351.2 Important Commands

xen-create-image

```
# create a pv image
xen-create-image \
  --hostname=lpic3-pv-guest \
  --memory=1gb \
  --vcpus=2 \
  --lvm=vg_xen \
  --bridge=xenbr0 \
  --dhcp \
  --pygrub \
  --password=vagrant \
  --dist=bookworm
```

xen-list-images

```
# list image
xen-list-image
```

xen-delete-image

```
# delete a pv image
xen-delete-image lpic3-pv-guest --lvm=vg_xen
```

xenstore-ls

```
# list xenstore infos
xenstore-ls
```

xl

```
# view xen information
xl infos

# list Domains
xl list
xl list lpic3-hvm-guest
xl list lpic3-hvm-guest -l

# uptime Domains
xl uptime

# pause Domain
xl pause 2
xl pause lpic3-hvm-guest

# save state Domains
xl -v save lpic3-hvm-guest ~root/image-lpic3-hvm-guest.save

# restore Domain
xl restore /root/image-lpic3-hvm-guest.save

# get Domain name
xl domname 2

# view dmesg information
xl dmesg

# monitoring domain
xl top
xentop
xen top

# Limit mem Dom0
xl mem-set 0 2048

# Limite cpu (not permanent after boot)
xl vcpu-set 0 2
```

```
# create DomainU - virtual machine
xl create /etc/xen/lpic3-pv-guest.cfg

# create DomainU virtual machine and connect to guest
xl create -c /etc/xen/lpic3-pv-guest.cfg

##----------------------------------------------
# create DomainU virtual machine HVM

## create logical volume
lvcreate -l +20%FREE -n lpic3-hvm-guest-disk  vg_xen

## create a ssh tunel for vnc
ssh -l vagrant -L 5900:localhost:5900  192.168.0.130

## configure /etc/xen/lpic3-hvm-guest.cfg
## set boot for cdrom: boot = "d"

## create domain hvm
xl create /etc/xen/lpic3-hvm-guest.cfg

## open vcn conection in your vnc client with localhost
## for view install details

## after installation finished, destroy domain: xl destroy <id_or_name>

## set /etc/xen/lpic3-hvm-guest.cfg: boot for hard disc: boot = "c"

## create domain hvm
xl create /etc/xen/lpic3-hvm-guest.cfg

## access domain hvm
xl console <id_or_name>
##----------------------------------------------

# connect in domain guest
xl console <id>|<name> (press enter)
xl console 1
xl console lpic3-pv-guest

#How do I exit domU "xl console" session
#Press ctrl+] or if you're using Putty press ctrl+5.

# Poweroff domain
xl shutdown lpic3-pv-guest

# destroy domain
xl destroy lpic3-pv-guest

# reboot domain
xl reboot lpic3-pv-guest

# list block devices
xl block-list 1
```

```
xl block-list lpic3-pv-guest

# detach block devices
xl block-detach lpic3-hvm-guest hdc
xl block-detach 2 xvdc

# attach block devices

## hard disk devices
xl block-attach lpic3-hvm-guest-ubuntu 'phy:/dev/vg_xen/lpic3-hvm-guest-disk2,xvde,w

## cdrom
xl block-attach lpic3-hvm-guest 'file:/home/vagrant/isos/ubuntu/seed.iso,xvdc:cdrom,
xl block-attach 2 'file:/home/vagrant/isos/ubuntu/seed.iso,xvdc:cdrom,r'

# insert and eject cdrom devices
xl cd-insert lpic3-hvm-guest-ubuntu xvdb  /home/vagrant/isos/ubuntu/ubuntu-24.04.1-l
xl cd-eject lpic3-hvm-guest-ubuntu xvdb
```

251.2 Notes

vif

In Xen, “vif” stands for Virtual Interface and is used to configure networking for virtual machines (domains).

By specifying “vif” directives in the domain configuration files, administrators can define network interfaces, assign IP addresses, set up VLANs, and configure other networking parameters for virtual machines running on Xen hosts. For example: vif = [‘bridge=xenbr0’], in this case, it connects the VM’s network interface to the Xen bridge named “xenbr0”.

```
<p align="right">(<a href="#topic-351.2">back to sub Topic 351.2</a>)</p>
<p align="right">(<a href="#topic-351">back to Topic 351</a>)</p>
<p align="right">(<a href="#readme-top">back to top</a>)</p>

---

<a name="topic-351.3"></a>

### 351.3 QEMU

![xen-kvm-qemu](/images/xen-kvm-qemu.png)

**Weight:** 4

**Description:** Candidates should be able to install, configure, maintain, migrate

**Key Knowledge Areas:**

* Understand the architecture of QEMU, including KVM, networking and storage
```

```
* Start QEMU instances from the command line
* Manage snapshots using the QEMU monitor
* Install the QEMU Guest Agent and VirtIO device drivers
* Troubleshoot QEMU installations, including networking and storage
* Awareness of important QEMU configuration parameters

#### 351.3 Cited Objects

```sh
Kernel modules: kvm, kvm-intel and kvm-amd
/dev/kvm
QEMU monitor
qemu
qemu-system-x86_64
ip
brctl
tunctl
```

351.3 Important Commands

351.3 Others Commands

check kvm module

```
# check if kvm is enabled
egrep -o '(vmx|svm)' /proc/cpuinfo
lscpu |grep Virtualization
lsmod|grep kvm
ls -l /dev/kvm
hostnamectl
systemd-detect-virt
```

```
# check if kvm is enabled
egrep -o '(vmx|svm)' /proc/cpuinfo
lscpu |grep Virtualization
lsmod|grep kvm
ls -l /dev/kvm

# check kernel infos
uname -a

# check root device
findmnt /

# mount a qcow2 image
## Example 1:
mkdir -p /mnt/qemu
guestmount -a os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2 -i /mnt/qemu/

## Example 2:
sudo guestfish --rw -a os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2
```
```

```
run
list-filesystems

# run commands in qcow2 images
## Example 1:
virt-customize -a  os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2  --run-command 'e
## Example 2:
sudo virt-customize -a os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2 \
  --run-command 'echo -e "auto ens3\niface ens3 inet dhcp" > /etc/network/interfaces

# generate mac
printf 'DE:AD:BE:EF:%02X:%02X\n' $((RANDOM%256)) $((RANDOM%256))
```

ip

```
# list links
ip link show

# create bridge
ip link add br0 type bridge
```

brctl

```
# list links
ip link show

# create bridge
ip link add br0 type bridge
```

qemu-img

```
# create image
qemu-img create -f qcow2 vm-disk-debian-12.qcow2 20G

# convert vmdk to qcow2 image
qemu-img convert \
  -f vmdk \
  -O qcow2 os-images/Debian_12.0.0_VMM/Debian_12.0.0_VMM_LinuxVMImages.COM.vmdk os-i
  -p \
  -m16

# check image
qemu-img info os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2
```

qemu-system-x86_64

```
# create vm with ISO
qemu-system-x86_64 \
```

```
  -name lpic3-debian-12 \
  -enable-kvm -hda vm-disk-debian-12.qcow2 \
  -cdrom /home/vagrant/isos/debian/debian-12.8.0-amd64-DVD-1.iso  \
  -boot d \
  -m 2048 \
  -smp cpus=2 \
  -k pt-br

# create vm with ISO using vnc in no gui servers \ ssh connections

## create ssh tunel in host
 ssh -l vagrant -L 5902:localhost:5902  192.168.0.131

## create vm
qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -k pt-br \
  -vnc :2 \
  -device qemu-xhci \
  -device usb-tablet \
  -device ide-cd,bus=ide.1,drive=cdrom,bootindex=1 \
  -drive id=cdrom,media=cdrom,if=none,file=/home/vagrant/isos/debian/debian-12.8.0-a
  -hda vm-disk-debian-12.qcow2 \
  -boot order=d \
  -vga std \
  -display none \
  -monitor stdio

# create vm with OS Image - qcow2

## create vm
qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -k pt-br \
  -vnc :2 \
  -hda os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2

## create vm with custom kernel params
qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -kernel /vmlinuz \
  -initrd /initrd.img \
  -append "root=/dev/mapper/debian--vg-root ro fastboot console=ttyS0" \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -k pt-br \
  -vnc :2 \
  -hda os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2
```

```
## create vm with and attach disk
qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -vnc :2 \
  -hda os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2 \
  -hdb vmdisk-debian12.qcow2 \
  -drive file=vmdisk-extra-debian12.qcow2,index=2,media=disk,if=ide \
  -netdev bridge,id=net0,br=qemubr0 \
  -device virtio-net-pci,netdev=net0

## create vm network netdev user
qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -vnc :2 \
  -hda os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2 \
  -netdev user,id=mynet0,net=192.168.0.150/24,dhcpstart=192.168.0.155,hostfwd=tcp::2
  -device virtio-net-pci,netdev=mynet0

## create vm network netdev tap (Private Network)
ip link add br0 type bridge ; ifconfig br0 up
qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -vnc :2 \
  -hda os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2 \
  -netdev tap,id=br0 \
  -device e1000,netdev=br0,mac=DE:AD:BE:EF:1A:24

## create vm with public bridge
#create a public bridge : https://www.linux-kvm.org/page/Networking

qemu-system-x86_64 \
  -name lpic3-debian-12 \
  -enable-kvm \
  -m 2048 \
  -smp cpus=2 \
  -hda os-images/Debian_12.0.0_VMM/Debian_12.0.0.qcow2 \
  -k pt-br \
  -vnc :2 \
  -device qemu-xhci \
  -device usb-tablet \
  -vga std \
  -display none \
  -netdev bridge,id=net0,br=qemubr0 \
  -device virtio-net-pci,netdev=net0
```

```
## get a ipv4 ip - open ssh in vm and:
dhcpclient ens4
```

QEMU Monitor

For initiate QEMU monitor in commandline use **-monitor stdio** param in **qemu-system-x86_64**

```
qemu-system-x86_64 -monitor stdio
```

Exit qemu-monitor:

```
ctrl+alt+2
```

```
# Managment
info status # vm info
info cpus # cpu information
info network # network informations
stop # pause vm
cont # start vm in status pause
system_powerdown # poweroff vm
system_reset # restart monitor


# Blocks
info block # block info
boot_set d # force boot iso
change ide1-cd0  /home/vagrant/isos/debian/debian-12.8.0-amd64-DVD-1.iso  # attach c
eject ide1-cd0 # detach cdrom

# Snapshots
info snapshots # list snapshots
savevm snapshot-01  # create snapshot
loadvm snapshot-01 # restore snapshot
delvm snapshot-01
```

Guest Agent

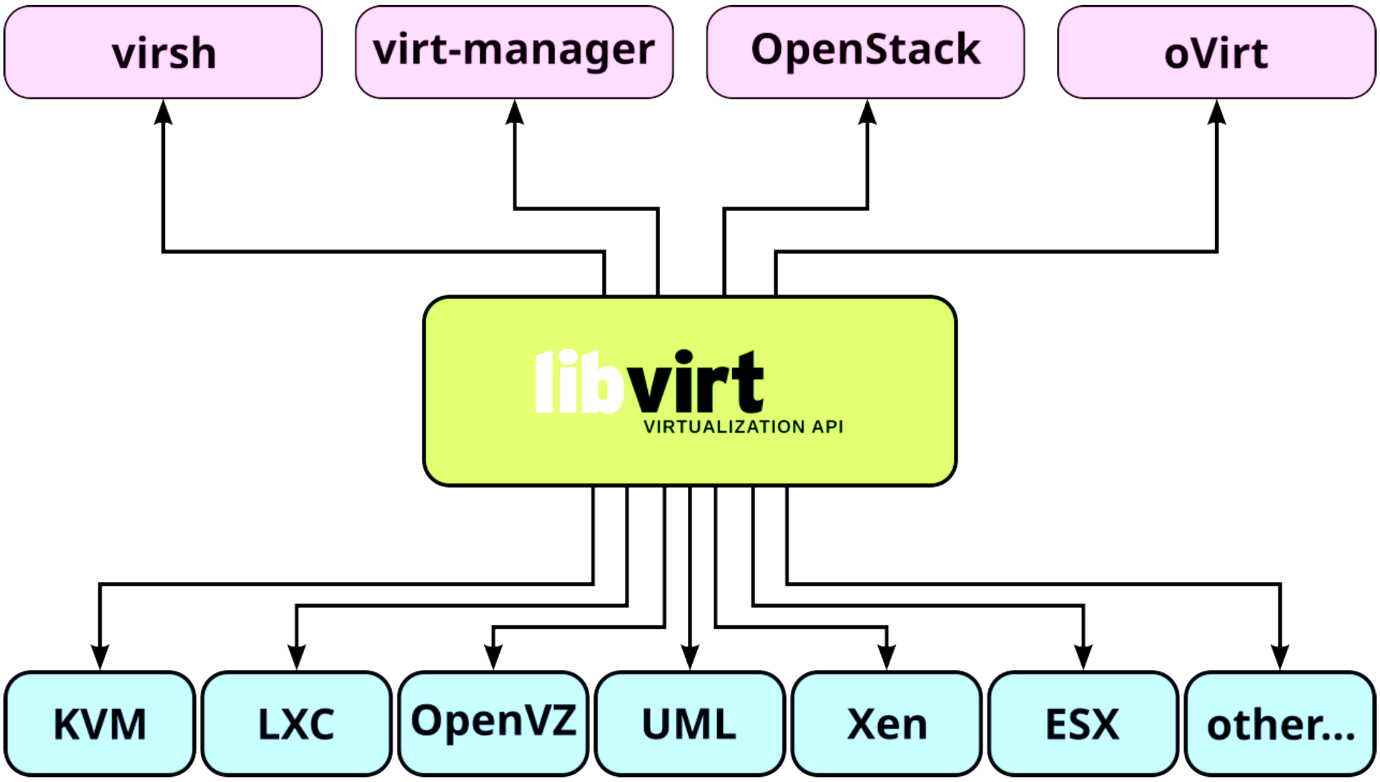
For enable, use:

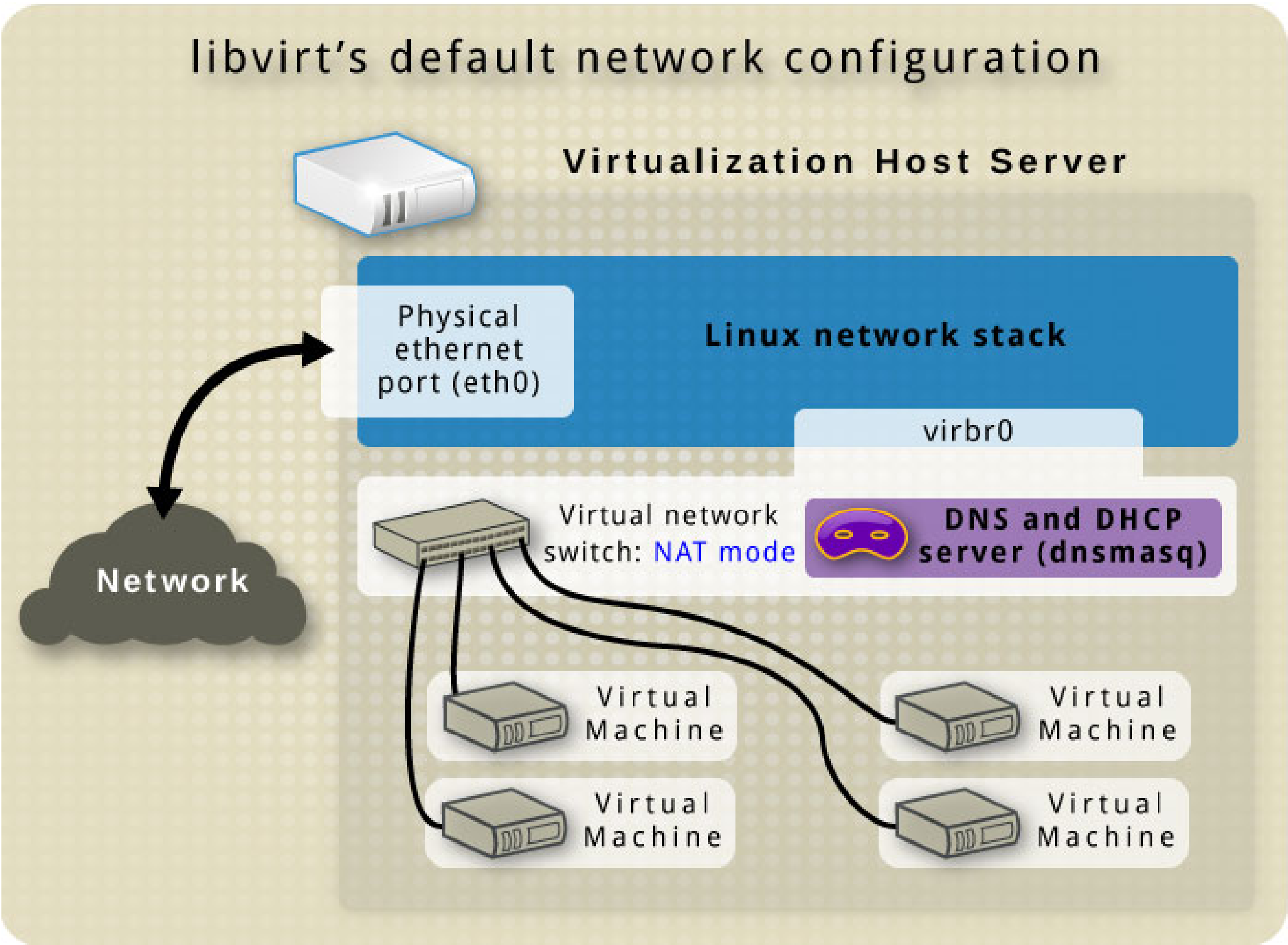
```
qemu-system-x86_x64
 -chardev socket,path=/tmp/qga.sock,server=on,wait=off,id=qga0 \
 -device virtio-serial \
 -device virtserialport,chardev=qga0,name=org.qemu.guest_agent.0
```

(back to sub Topic 351.3)

(back to Topic 351)

351.4 Libvirt Virtual Machine Management





Weight: 9

Description: Candidates should be able to manage virtualization hosts and virtual machines (‘libvirt domains’) using libvirt and related tools.

Key Knowledge Areas:

- Understand the architecture of libvirt
- Manage libvirt connections and nodes
- Create and manage QEMU and Xen domains, including snapshots
- Manage and analyze resource consumption of domains
- Create and manage storage pools and volumes
- Create and manage virtual networks
- Migrate domains between nodes
- Understand how libvirt interacts with Xen and QEMU
- Understand how libvirt interacts with network services such as dnsmasq and radvd
- Understand libvirt XML configuration files
- Awareness of virtlogd and virtlockd

351.4 Cited Objects

```
libvirtd
/etc/libvirt/
/var/lib/libvirt
/var/log/libvirt
virsh (including relevant subcommands)
```

351.4 Important Commands

virsh

```
# using env variable for set virsh uri (local or remotly)
export LIBVIRT_DEFAULT_URI=qemu:///system
export LIBVIRT_DEFAULT_URI=xen+ssh://vagrant@192.168.0.130
export LIBVIRT_DEFAULT_URI='xen+ssh://vagrant@192.168.0.130?keyfile=/home/vagrant/.s

# COMMONS

# get helps
virsh help
virsh help pool-create

# view version
virsh version

# view system info
```

Light

```
sudo virsh sysinfo

# view node info
virsh nodeinfo

# hostname
virsh hostname

# check vcn allocated port
virsh vncdisplay <domain_id>
virsh vncdisplay <domain_name>
virsh vncdisplay rocky9-server01

# HYPERVISIONER

# view libvirt hypervisioner connection
virsh uri

# list valid hypervisioners
virt-host-validate
virt-host-validate qemu

# test connetion uri(vm test)
virsh -c test:///default list

# connect remotly
virsh -c xen+ssh://vagrant@192.168.0.130
virsh -c xen+ssh://vagrant@192.168.0.130 list
virsh -c qemu+ssh://vagrant@192.168.0.130/system list

# connect remotly without enter password
virsh -c 'xen+ssh://vagrant@192.168.0.130?keyfile=/home/vagrant/.ssh/skynet-key-ecds

# STORAGE

# list storage pools
virsh pool-list --details

# list all storage pool
virsh pool-list --all --details

# get a pool configuration
virsh pool-dumpxml default

# get pool info
virsh pool-info default

# create a storage pool
virsh pool-define-as --name default --type dir --target /var/lib/libvirt/images

# create a storage pool with dumpxml
virsh pool-create --overwrite --file configs/kvm/libvirt/pool.xml

# start storage pool
virsh pool-start default
```

```
# set storage pool for autostart
virsh pool-autostart default

# stop storage pool
virsh pool-destroy linux

# delete xml storage pool file
virsh pool-undefine linux

# edit storage pool
virsh pool-edit linux

# list volumes
virsh vol-list linux

# get volume infos
virsh vol-info Debian_12.0.0.qcow2 os-images
virsh vol-info --pool os-images Debian_12.0.0.qcow2

# get volume xml
virsh vol-dumpxml rocky9-disk1 default

# create volume
virsh vol-create-as default --format qcow2 disk1 10G

# delete volume
virsh vol-delete  disk1 default

# DOMAINS \ INSTANCES \ VIRTUAL MACHINES

# list domain\instance\vm
virsh list
virsh list --all

# create domain\instance\vm
virsh create configs/kvm/libvirt/rocky9-server03.xml

# view domain\instance\vm info
virsh dominfo rocky9-server01

# view domain\instance\vm xml
virsh dumpxml rocky9-server01

# edit domain\instance\vm xml
virsh edit rocky9-server01

# stop domain\instance\vm
virsh shutdown rocky9-server01 # gracefully
virsh destroy 1
virsh destroy rocky9-server01

# suspend domain\instance\vm
virsh suspend rocky9-server01
```

```
# resume domain\instance\vm
virsh resume rocky9-server01

# start domain\instance\vm
virsh start rocky9-server01

# remove domain\instance\vm
virsh undefine rocky9-server01

# remove domain\instance\vm and storage volumes
virsh undefine rocky9-server01 --remove-all-storage

# save domain\instance\vm
virsh save rocky9-server01 rocky9-server01.qcow2

# restore domain\instance\vm
virsh restore rocky9-server01.qcow2

# list snapshots
virsh snapshot-list rocky9-server01

# create snapshot
virsh snapshot-create rocky9-server01

# restore snapshot
virsh snapshot-revert rocky9-server01 1748983520

# view snapshot xml
virsh snapshot-info rocky9-server01 1748983520

# dumpxml snapshot
virsh snapshot-dumpxml rocky9-server01 1748983520

# xml snapshot path
/var/lib/libvirt/qemu/snapshot/rocky9-server01/

# view snapshot info
virsh snapshot-info rocky9-server01 1748983671

# edit snapshot
virsh snapshot-edit rocky9-server01 1748983520

# delete snapshot
virsh snapshot-delete rocky9-server01 1748983520

# DEVICES

# list block devices
virsh domblklist rocky9-server01 --details

# add cdrom media
virsh change-media rocky9-server01 sda /home/vagrant/isos/rocky/Rocky-9.5-x86_64-min
virsh attach-disk rocky9-server01 /home/vagrant/isos/rocky/Rocky-9.5-x86_64-minimal.

# remove cdrom media
```

Light

```
virsh change-media rocky9-server01 sda --eject

# add new disk
virsh attach-disk rocky9-server01  /var/lib/libvirt/images/rocky9-disk2  vdb --persi

# remove disk
virsh detach-disk rocky9-server01 vdb --persistent

# RESOURCES (CPU and Memory)

# get cpu infos
virsh vcpuinfo rocky9-server01 --pretty
virsh dominfo rocky9-server01 | grep 'CPU'

# get vcpu count
virsh vcpucount rocky9-server01

# set vcpus maximum config
virsh setvcpus rocky9-server01 --count 4 --maximum --config
virsh shutdown rocky9-server01
virsh start rocky9-server01

# set vcpu current config
virsh setvcpus rocky9-server01 --count 4 --config

# set vcpu current live
virsh setvcpus rocky9-server01 --count 3 --current
virsh setvcpus rocky9-server01 --count 3 --live

# configure vcpu afinity config
virsh vcpupin rocky9-server01 0 7 --config
virsh vcpupin rocky9-server01 1 5-6 --config

# configure vcpu afinity current
virsh vcpupin rocky9-server01 0 7
virsh vcpupin rocky9-server01 1 5-6

# set maximum memory config
virsh setmaxmem rocky9-server01 3000000 --config
virsh shutdown rocky9-server01
virsh start rocky9-server01

# set current memory config
virsh setmem rocky9-server01 2500000 --current

# NETWORK

# get netwwork bridges
brctl show

# get iptables rules for libvirt
sudo iptables -L -n -t  nat

# list network
virsh net-list --all
```

```
# set default network
virsh net-define /etc/libvirt/qemu/networks/default.xml

# get network infos
virsh net-info default

# get xml network
virsh net-dumpxml default

# xml file
cat /etc/libvirt/qemu/networks/default.xml

# dhcp config
sudo cat /etc/libvirt/qemu/networks/default.xml | grep -A 10 dhcp
sudo cat /var/lib/libvirt/dnsmasq/default.conf

# get domain ipp address
virsh net-dhcp-leases default
virsh net-dhcp-leases default --mac 52\:54\:00\:89\:19\:86

# edit network
virsh net-edit default

# get domain network detais
virsh domiflist debian-server01

# path for network filter files
/etc/libvirt/nwfilter/

# list network filters
virsh nwfilter-list

# create network filter - block icmp traffic
virsh nwfilter-define block-icmp.xml
# virsh edit Debian-Server
    #  <interface type='network'>
    #        ...
    #        <filterref filter='block-icmp'/>
    #        ...
    # </interface>
# virsh destroy debian-server01
# virsh start debian-server01

# delete network filter
virsh nwfilter-undefine block-icmp

# get xml network filter
virsh nwfilter-dumpxml block-icmp
```

virt-install

```
# list os variants
virt-install --os-variant list
osinfo-query os

# create domain\instance\vm with iso file
virsh vol-create-as default --format qcow2 rocky9-disk1 20G
virt-install --name rocky9-server01 \
--vcpus 2 \
--cpu host \
--memory 2048 \
--disk vol=default/rocky9-disk1 \
--cdrom /home/vagrant/isos/rocky/Rocky-9.5-x86_64-minimal.iso \
--os-variant=rocky9 \
--graphics vnc,listen=0.0.0.0,port=5905

# create debian domain\instance\vm with qcow2 file
virt-install --name debian-server01 \
--vcpus 2 \
--ram 2048 \
--disk vol=os-images/Debian_12.0.0.qcow2 \
--import \
--osinfo detect=on \
--graphics vnc,listen=0.0.0.0,port=5906 \
--network network=default \
--noautoconsole

# create rocky9 domain\instance\vm with qcow2 file
virt-install --name rocky9-server02 \
--vcpus 2 \
--ram 2048 \
--disk path=os-images/RockyLinux_9.4_VMG/RockyLinux_9.4.qcow2,format=qcow2,bus=virti
--import \
--osinfo detect=on \
--graphics vnc,listen=0.0.0.0,port=5907 \
--network bridge=qemubr0,model=virtio \
--noautoconsole

# open domain\instance\vm gui console
virt-viewer debian-server01

# check metadata domain\instance\vm file (if uri is qemu:////system)
less /etc/libvirt/qemu/debian-server01.xml
```

(back to sub Topic 351.4)

(back to Topic 351)

(back to top)

351.5 Virtual Machine Disk Image Management

 Light

Virtual Machine Disk Images

Format	Hypervisor	Size	Compression	Encryption	Snapchots
IMG	Generic	Gyramic	No	No	No
RAW	Generico	Hyper-V	Yes	No	No
VHD	Hyper-V (VMWare, VirtualBox)	Dynamic	Yes	No	Sim
VMDK	Hyper-V	Dynamic	Yes	Yes	Sim
VDI	VirtualBox	Dynamic	Yes	Yes	Sim
QCOW2	QEMU, KVM	Dynamic	Yes	Yes	Sim

Weight: 3

Description: Candidates should be able to manage virtual machines disk images. This includes converting disk images between various formats and hypervisors and accessing data stored within an image.

Key Knowledge Areas:

- Understand features of various virtual disk image formats, such as raw images, qcow2 and VMDK
- Manage virtual machine disk images using qemu-img
- Mount partitions and access files contained in virtual machine disk images using libguestfish
- Copy physical disk content to a virtual machine disk image
- Migrate disk content between various virtual machine disk image formats
- Awareness of Open Virtualization Format (OVF)

351.5 Cited Objects

```
qemu-img
guestfish (including relevant subcommands)
guestmount
guestumount
virt-cat
virt-copy-in
virt-copy-out
```

```
virt-diff
virt-inspector
virt-filesystems
virt-rescue
virt-df
virt-sparsify
virt-p2v
virt-p2v-make-disk
virt-v2v
```

351.5 Important Commands

351.5.1 qemu-img

```
# Display detailed information about a disk image
qemu-img info UbuntuServer_24.04.qcow2

# Create a new 22G raw disk image (default format is raw)
qemu-img create new-disk 22G

# Create a new 22G disk image in qcow2 format
qemu-img create -f qcow2 new-disk2 22G

# Convert a VDI image to raw format using 5 threads and show progress
qemu-img convert -f vdi -O raw Ubuntu-Server.vdk new-Ubuntu.raw -m5 -p

# Convert vmdk to qcow2 image
qemu-img convert \
-f vmdk \
-O qcow2 os-images/UbuntuServer_24.04_VM/UbuntuServer_24.04_VM_LinuxVMImages.COM.vmd
os-images/UbuntuServer_24.04_VM/UbuntuServer_24.04.qcow2 \
-p \
-m16

# Resize a raw image to 30G
qemu-img resize -f raw new-disk 30G

# Resize a qcow2 image to 15G(actual size 30Gdisk 30G)
qemu-img resize -f raw --shrink new-disk 15G

# Snapshots

# List all snapshots in the image
qemu-img snapshot -l new-disk2.qcow2

# Create a snapshot named SNAP1
qemu-img snapshot -c SNAP1 disk

# Apply a snapshot by ID or name
qemu-img snapshot -a 123456789 disk

# Delete the snapshot named SNAP1
qemu-img snapshot -d SNAP1 disk
```

 Light

guestfish

```
# set enviroment variables for guestfish
export LIBGUESTFS_BACKEND_SETTINGS=force_tcg

# Launch guestfish with a disk image
guestfish -a UbuntuServer_24.04.qcow2
#run
#list-partitions

# Run the commands in a script file
guestfish -a UbuntuServer_24.04.qcow2 -m /dev/sda -i < script.ssh

# Interactively run commands
guestfish --rw -a UbuntuServer_24.04.qcow2 <<'EOF'
run
list-filesystems
EOF

# Copy a file from the guest image to the host
export LIBGUESTFS_BACKEND_SETTINGS=force_tcg
sudo guestfish --rw -a UbuntuServer_24.04.qcow2 -i <<'EOF'
copy-out /etc/hostname /tmp/
EOF

# Copy a file from the host into the guest image
echo "new-hostname" > /tmp/hostname
export LIBGUESTFS_BACKEND_SETTINGS=force_tcg
sudo guestfish --rw -a UbuntuServer_24.04.qcow2 -i <<'EOF'
copy-in /tmp/hostname /etc/
EOF

# View contents of a file in the guest image
guestfish --ro -a UbuntuServer_24.04.qcow2 -i <<'EOF'
cat /etc/hostname
EOF

# List files in the guest image
export LIBGUESTFS_BACKEND_SETTINGS=force_tcg
guestfish --rw -a UbuntuServer_24.04.qcow2 -i <<'EOF'
ls /home/ubuntu
EOF

# Edit a file in the guest image
export LIBGUESTFS_BACKEND_SETTINGS=force_tcg
guestfish --rw -a UbuntuServer_24.04.qcow2 -i <<'EOF'
edit /etc/hosts
EOF
```

guestmount

```
# Mount a disk image to a directory
guestmount -a UbuntuServer_24.04.qcow2 -m /dev/ubuntu-vg/ubuntu-lv /mnt/ubuntu
# domain
guestmount -d rocky9-server02 -m /dev/ubuntu-vg/ubuntu-lv /mnt/ubuntu

# Mount a specific partition from a disk image
guestmount -a UbuntuServer_24.04.qcow2 -m /dev/sda2 /mnt/ubuntu
# domain
guestmount -d debian-server01 --ro -m  /dev/debian-vg/root /mnt/debian
```

guestumount

```
# Umount a disk image to a directory
sudo guestunmount /mnt/ubuntu
```

virt-df

```
# Show free and used space on virtual machine filesystems
virt-df UbuntuServer_24.04.qcow2 -h
virt-df -d rocky9-server02 -h
```

virt-filesystems

```
# List filesystems, partitions, and logical volumes in a VM disk image (disk image)
virt-filesystems -a UbuntuServer_24.04.qcow2 --all --long -h

# List filesystems, partitions, and logical volumes in a VM disk image (domain)
virt-filesystems -d debian-server01 --all --long -h
```

virt-inspector

```
# Inspect and report on the operating system in a VM disk image
virt-inspector -a UbuntuServer_24.04.qcow2 #(disk)
virt-inspector -d debian-server01 #(domain)
```

virt-cat

```
# Display the contents of a file inside a VM disk image
virt-cat -a UbuntuServer_24.04.qcow2 /etc/hosts
virt-cat -d debian-server01 /etc/hosts #(domain)
```

virt-diff

 Light

```
# Show differences between two VM disk images
virt-diff -a UbuntuServer_24.04.qcow2 -A Rocky-Linux.qcow2
```

virt-sparsify

```
# Make a VM disk image smaller by removing unused space
virt-sparsify UbuntuServer_24.04.qcow2 UbuntuServer_24.04-sparse.qcow2
```

virt-resize

```
# Resize a VM disk image or its partitions
virt-filesystems -a UbuntuServer_24.04.qcow2 --all --long -h #(check size of partiti
qemu-img create -f qcow2 UbuntuServer_24.04-expanded.qcow2 100G #(create new disk im
virt-resize --expand /dev/ubuntu-vg/ubuntu-lv \
UbuntuServer_24.04.qcow2 UbuntuServer_24.04-expanded.qcow2
```

virt-copy-in

```
# Copy files from the host into a VM disk image

virt-copy-in -a UbuntuServer_24.04.qcow2 ~vagrant/test-virt-copy-in.txt /home/ubuntu
```

virt-copy-out

```
# Copy files from a VM disk image to the host
virt-copy-out -a UbuntuServer_24.04.qcow2 /home/ubuntu/.bashrc /tmp
```

virt-ls

```
# List files and directories inside a VM disk image
virt-ls -a UbuntuServer_24.04.qcow2 /home/ubuntu
```

virt-rescue

```
# Launch a rescue shell on a VM disk image for recovery
virt-rescue -a UbuntuServer_24.04.qcow2
```

virt-sysprep

```
# Prepare a VM disk image for cloning by removing system-specific data
virt-sysprep -a UbuntuServer_24.04.qcow2
```

virt-v2v

```
# Convert a VM from a foreign hypervisor to run on KVM
virt-v2v -i disk input-disk.img -o local -os /var/tmp
```

virt-p2v

```
# Convert a physical machine to use KVM
```

virt-p2v-make-disk

```
# Create a bootable disk image for physical to virtual conversion
sudo virt-p2v-make-disk -o output.img
```

351.5 Notes

OVF: Open Virtualization Format

OVF: An open format that defines a standard for packaging and distributing virtual machines across different environments.

The generated package has the .ova extension and contains the following files:

- .ovf: XML file with metadata defining the virtual machine environment
- Image files: .vmdk, .vhd, .vhdx, .qcow2, .raw
- Additional files: metadata, snapshots, configuration, hash

(back to sub Topic 351.5)

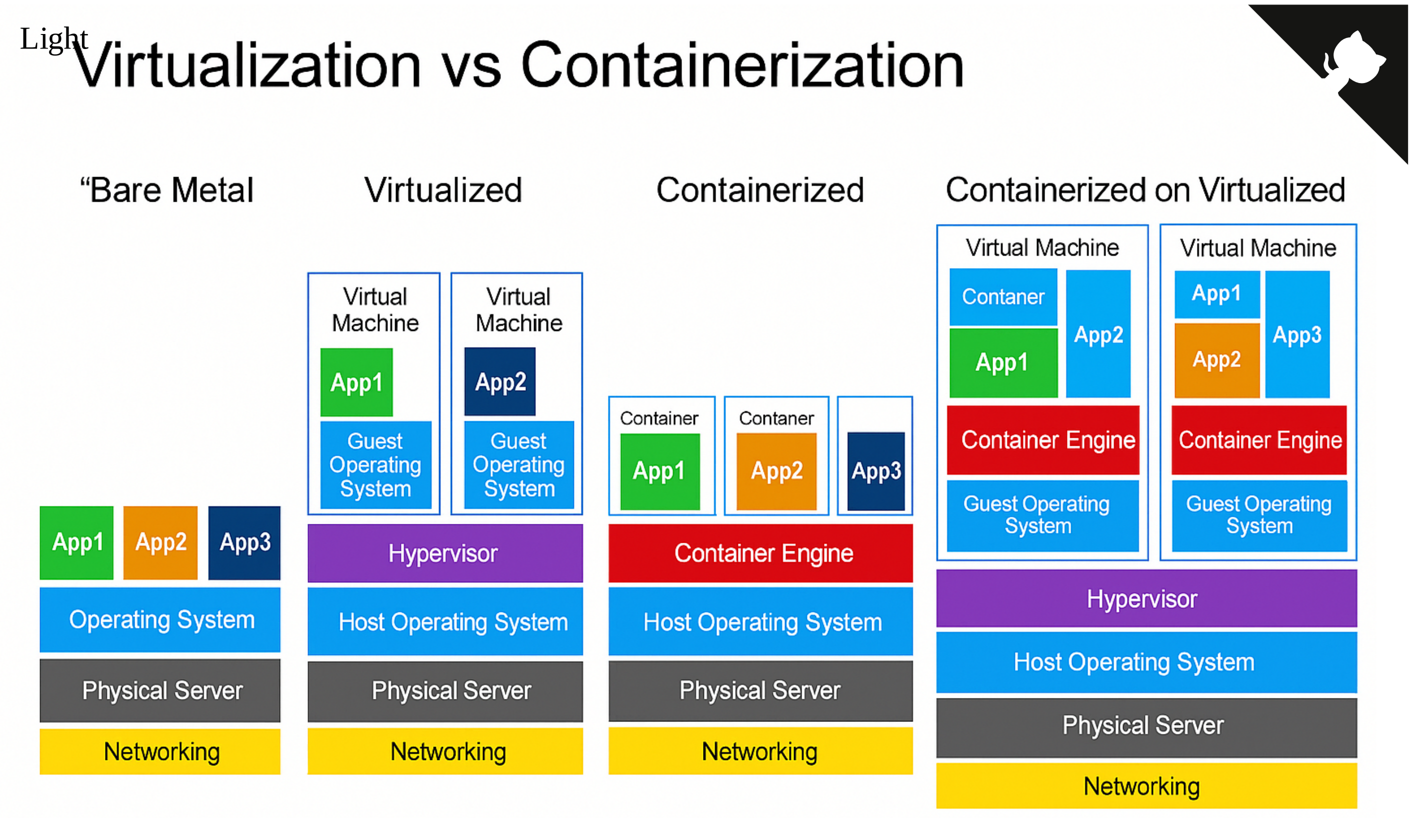
(back to Topic 351)

(back to top)

Topic 352: Container Virtualization

352.1 Container Virtualization Concepts

Light




```
timeline
    title Time Line Containers Evolution
    1979 : chroot
    2000 : FreeBSD Jails
    2002 : Linux Namespaces
    2005 : Solaris Containers
    2007 : cgroups
    2008 : LXC
    2013 : Docker
    2015 : Kubernetes
```

Weight: 7

Description: Candidates should understand the concept of container virtualization. This includes understanding the Linux components used to implement container virtualization as well as using standard Linux tools to troubleshoot these components.

Key Knowledge Areas:

- Understand the concepts of system and application container
- Understand and analyze kernel namespaces
- Understand and analyze control groups
- Understand and analyze capabilities

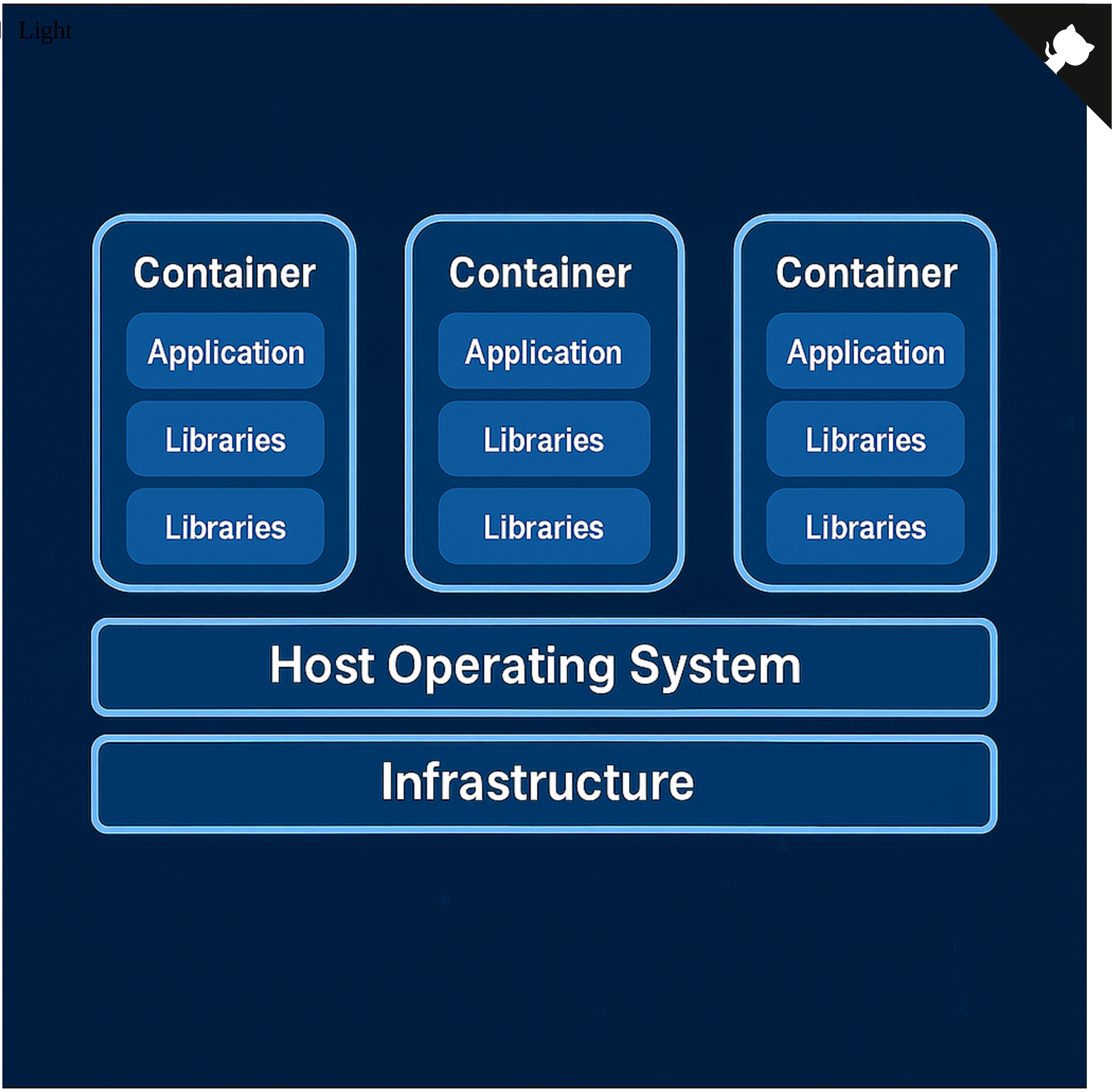
- Understand the role of seccomp, SELinux and AppArmor for container virtualization
- Understand how LXC and Docker leverage namespaces, cgroups, capabilities, seccomp and MAC
- Understand the principle of runc
- Understand the principle of CRI-O and containerd
- Awareness of the OCI runtime and image specifications
- Awareness of the Kubernetes Container Runtime Interface (CRI)
- Awareness of podman, buildah and skopeo
- Awareness of other container virtualization approaches in Linux and other free operating systems, such as rkt, OpenVZ, systemd-nspawn or BSD Jails

352.1 Cited Objects

```
nsenter
unshare
ip (including relevant subcommands)
capsh
/sys/fs/cgroups
/proc/[0-9]+/ns
/proc/[0-9]+/status
```

Understanding Containers

 Light



Containers are a lightweight virtualization technology that package applications along with their required dependencies — code, libraries, environment variables, and configuration files — into isolated, portable, and reproducible units.

> In simple terms: a container is a self-contained box that runs your application the same way, anywhere.

What Is a Container?

Unlike Virtual Machines (VMs), containers do not virtualize hardware. Instead, they virtualize the operating system. Containers share the same Linux kernel with the host, but each one operates in a fully isolated user space.

 Containers vs Virtual Machines:

Light

Feature	Containers	Virtual Machines
OS Kernel	Shared with host	Each VM has its own OS
Startup time	Fast (seconds or less)	Slow (minutes)
Image size	Lightweight (MBs)	Heavy (GBs)
Resource efficiency	High	Lower
Isolation mechanism	Kernel features (namespaces)	Hypervisor

Key Characteristics of Containers

- **Lightweight**: Share the host OS kernel, reducing overhead and enabling fast startup.
- **Portable**: Run consistently across different environments (dev, staging, prod, cloud, on-prem).
- **Isolated**: Use namespaces for process, network, and filesystem isolation.
- **Efficient**: Enable higher density and better resource utilization than traditional VMs.
- **Scalable**: Perfect fit for microservices and cloud-native architecture.

Types of Containers

1. System Containers
   - Designed to run the entire OS, Resemble virtual machines.
   - Support multiple processes and system services (init, syslog).
   - Ideal for legacy or monolithic applications.
   - Example: LXC, libvirt-lxc.
2. Application Containers
   - Designed to run a single process.
   - Stateless, ephemeral, and horizontally scalable.
   - Used widely in modern DevOps and Kubernetes environments.
   - Example: Docker, containerd, CRI-O.

Popular Container Runtimes

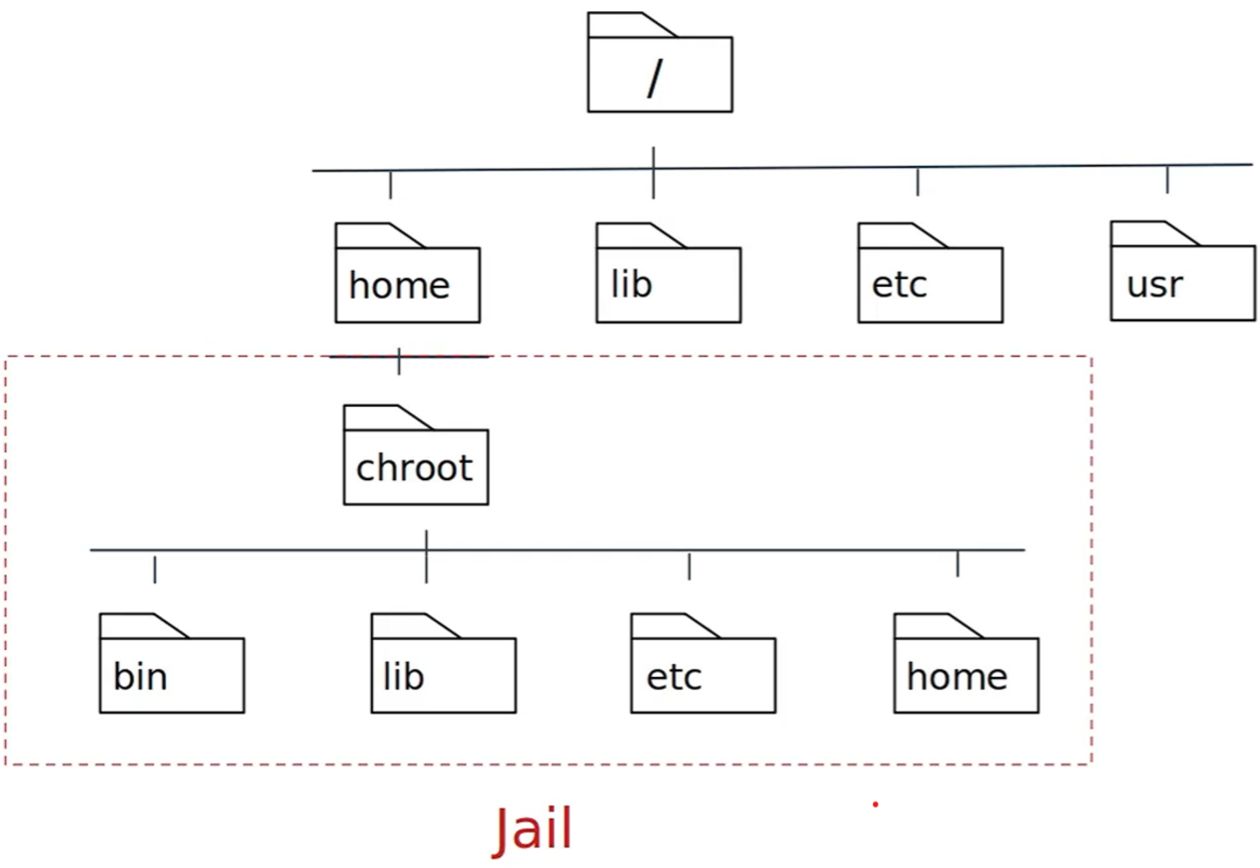
Runtime	Description
Docker	Most widely adopted CLI/daemon for building and running containers.
containerd	Lightweight runtime powering Docker and Kubernetes.
CRI-O	Kubernetes-native runtime for OCI containers.

Runtime	Description
LXC	Traditional Linux system containers, closer to full OS.
RKT	Security-focused runtime (deprecated).

Container Internals and Security Elements

Component	Role
Namespaces	Isolate processes, users, mounts, networks.
cgroups	Control and limit resource usage (CPU, memory, IO).
Capabilities	Fine-grained privilege control inside containers.
seccomp	Restricts allowed syscalls to reduce attack surface.
AppArmor / SELinux	Mandatory Access Control enforcement at kernel level.

Understanding chroot - Change Root Directory in Unix/Linux



What is chroot?

chroot (short for change root) is a system call and command on Unix-like operating systems changes the apparent root directory (/) for the current running process and its children. This creates an isolated environment, commonly referred to as a chroot jail.

Purpose and Use Cases

- Isolate applications for security (jailing).
- Create testing environments without impacting the rest of the system.
- System recovery (e.g., boot into LiveCD and chroot into installed system).
- Building software packages in a controlled environment.

Minimum Required Structure

The chroot environment must have its own essential files and structure:

```
/mnt/myenv/
├── bin/
│   └── bash
├── etc/
├── lib/
├── lib64/
├── usr/
├── dev/
├── proc/
└── tmp/
```

Use ldd to identify required libraries:

```
ldd /bin/bash
```

Limitations and Security Considerations

- chroot is not a security boundary like containers or VMs.
- A privileged user (root) inside the jail can potentially break out.
- No isolation of process namespaces, devices, or kernel-level resources.

For stronger isolation, consider alternatives like:

- Linux containers (LXC, Docker)
- Virtual machines (KVM, QEMU)
- Kernel namespaces and cgroups

Test chroot with debootstrap

```
# download debain files
sudo debootstrap stable ~vagrant/debian http://deb.debian.org/debian
```

Light

```
sudo chroot ~vagrant/debian bash
```

:🧪 Lab chroot

Use this script for lab: chroot.sh

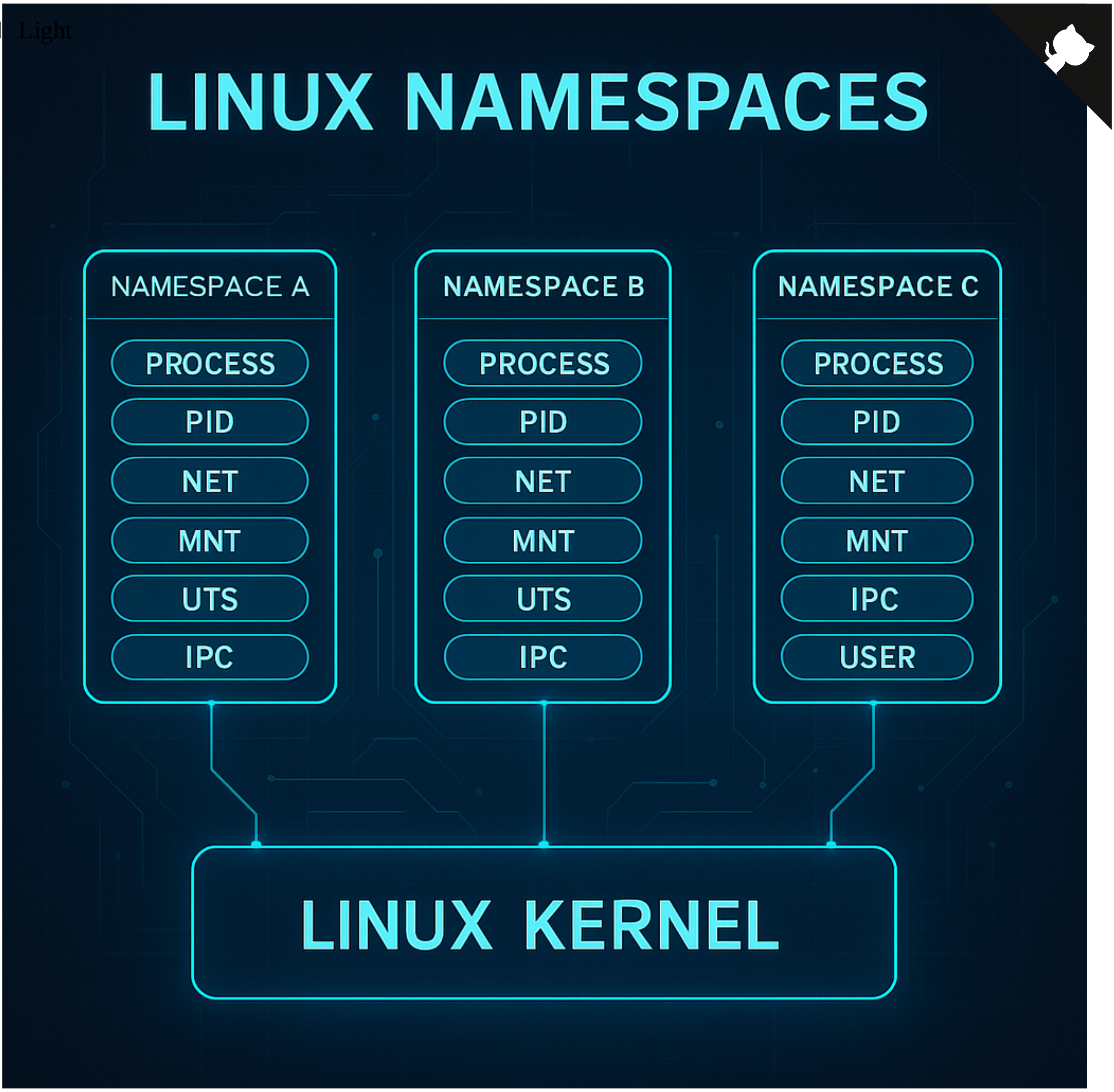
Output:

```
======================
📊 Chroot Test Summary
======================
[SUMMARY]
[CHROOT UTS] 🌐 Hostname: lpic3-topic-352-container
[CHROOT USER] 👤 UID/GID: uid=0(root) gid=0(root) groups=0(root)
[CHROOT PID] 🧵 Running processes:
    USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
    root           1  0.0  0.0 168204 12428 ?        Ss   01:06   0:01 /sbin/init
    root           2  0.0  0.0      0     0 ?        S    01:06   0:00 [kthreadd]
    root           3  0.0  0.0      0     0 ?        I<   01:06   0:00 [rcu_gp]
    root           4  0.0  0.0      0     0 ?        I<   01:06   0:00 [rcu_par_gp]
[CHROOT FS] 📁 Root filesystem:
bin chroot-lab-summary.sh dev etc lib lib64 proc sys tmp usr
[CHROOT MOUNT] 📂/tmp write: ✓ OK
======================

bash-5.2# ls /
bin  dev  etc  lib  lib64  proc  sys  tmp  usr
bash-5.2# ls /etc/
group  passwd  resolv.conf
bash-5.2# ls /bin
bash
bash-5.2# ls /tmp/
chroot-testfile
bash-5.2# 
```

🧠 Understanding Linux Namespaces

 Light



Namespaces are a core Linux kernel feature that enable process-level isolation. They create separate "views" of global system resources — such as process IDs, networking, filesystems, and users — so that each process group believes it is running in its own system.

> In simple terms: namespaces trick a process into thinking it owns the machine, even though it's just sharing it.

This is the foundation for container isolation.

What Do Namespaces Isolate?

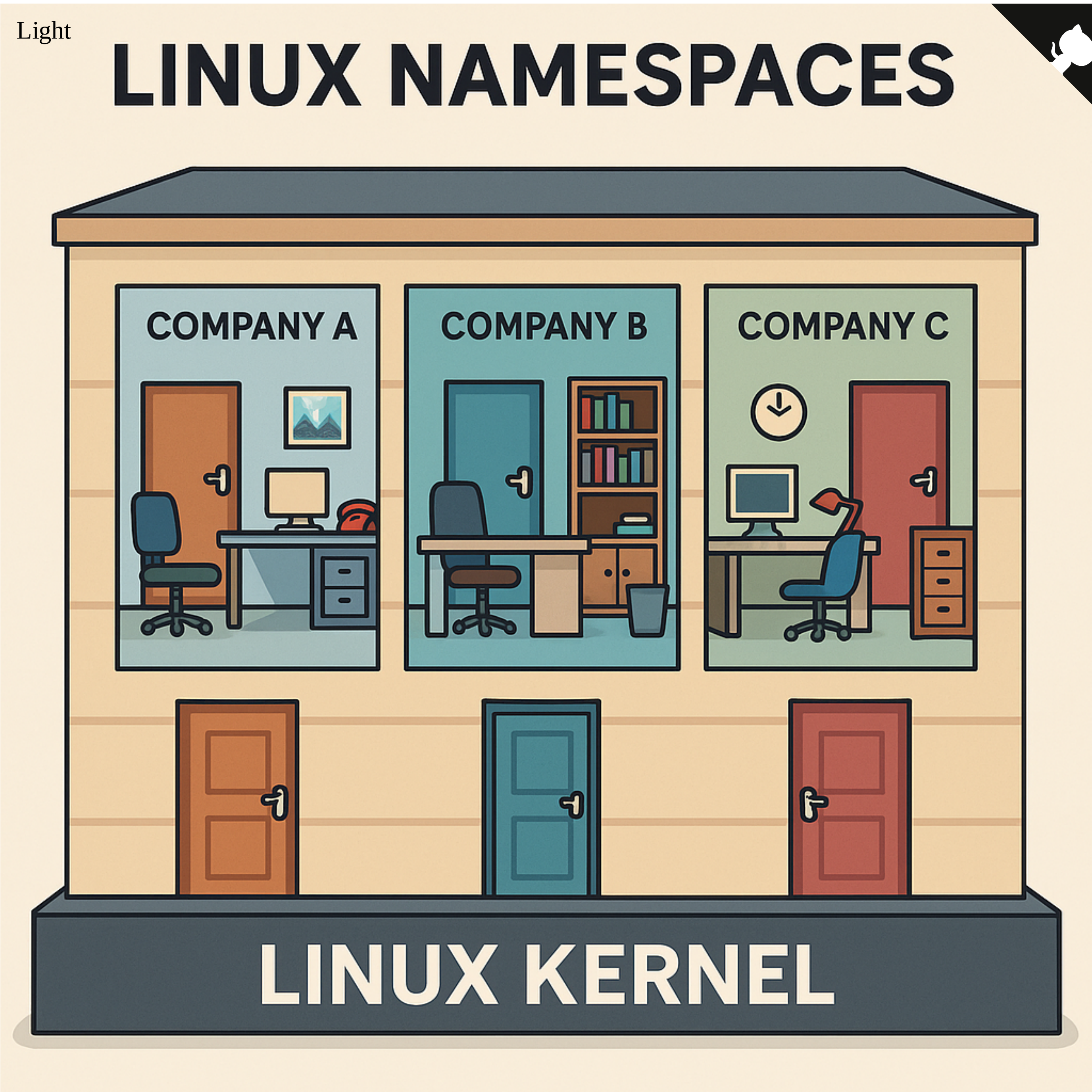
Each namespace type isolates a specific system resource. Together, they make up the sandbox that a container operates in:

Namespace	Isolates...	Real-world example

Namespace	Isolates...	Real-world example
PID	Process IDs	Processes inside a container see a different PID space
Mount	Filesystem mount points	Each container sees its own root filesystem
Network	Network stack	Containers have isolated IPs, interfaces, and routes
UTS	Hostname and domain name	Each container sets its own hostname
IPC	Shared memory and semaphores	Prevents inter-process communication between containers
User	User and group IDs	Enables fake root (UID 0) inside the container
Cgroup (v2)	Control group membership	Ties into resource controls like CPU and memory limits

Visual Analogy

Light



Imagine a shared office building:

- All tenants share the same foundation (Linux kernel).
- Each company has its own office (namespace): different locks, furniture, phone lines, and company name.
- To each tenant, it feels like their own building.

That's exactly how containers experience the system — isolated, yet efficient.

How Containers Use Namespaces

When you run a container (e.g., with Docker or Podman), the runtime creates a new set of namespaces:

```
docker run -it --rm alpine sh
```

This command gives the process:

- A new PID namespace → it's process 1 inside the container.
- A new network namespace → its own virtual Ethernet.
- A mount namespace → a container-specific root filesystem.
- Other namespaces depending on configuration (user, IPC, etc.)

The result: a lightweight, isolated runtime environment that behaves like a separate system.

Complementary Kernel Features

Namespaces hide resources from containers. But to control how much they can use and what they can do, we need additional mechanisms:

Cgroups (Control Groups)

Cgroups allow the kernel to limit, prioritize, and monitor resource usage across process groups.

Resource	Use case examples
CPU	Limit CPU time per container
Memory	Cap RAM usage
Disk I/O	Throttle read/write operations
Network (v2)	Bandwidth restrictions

Prevents the "noisy neighbor" problem by stopping one container from consuming all system resources.

Capabilities

Traditional Linux uses a binary privilege model: root (UID 0) can do everything, everyone else is limited.

Capability	Allows...
`CAP_NET_BIND_SERVICE`	Binding to privileged ports (e.g. 80, 443)
`CAP_SYS_ADMIN`	A powerful catch-all for system admin tasks
`CAP_KILL`	Sending signals to arbitrary processes

By dropping unnecessary capabilities, containers can run with only what they need — reducing risk.

Security Mechanisms

Used in conjunction with namespaces and cgroups to lock down what a containerized process can do:

Feature	Description
seccomp	Whitelist or block Linux system calls (syscalls)
AppArmor	Apply per-application security profiles
SELinux	Enforce Mandatory Access Control with tight system policies

Summary for Beginners

- Namespaces isolate what a container can see
- Cgroups control what it can use
- Capabilities and security modules define what it can do

Together, these kernel features form the technical backbone of container isolation — enabling high-density, secure, and efficient application deployment without full VMs.

Lab Namespaces

Use this script for lab: namespace.sh

Output:

Light

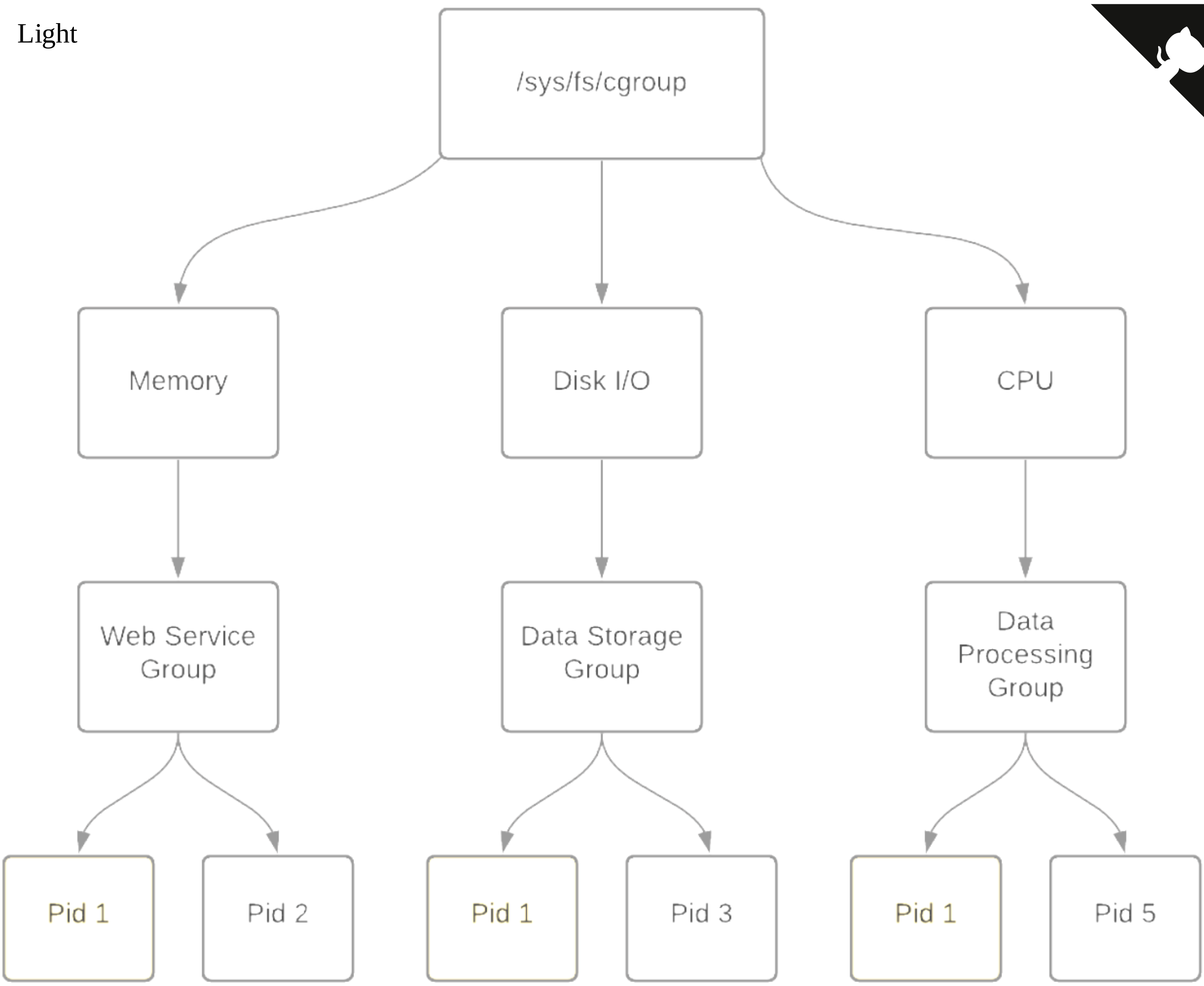
```
================================
  Namespace Simulator Lab
   By Silvestrini - 2025
================================
Choose an action:
1. Prepare minimal chroot for namespace lab
2. Run namespace lab (unshare + chroot + summary)
9. Teardown/Cleanup Temporary Files
0. Exit
================================
Enter your choice [0,1,2,9]: 2
[INFO] 💻 Launching unshare with full isolation 🚀
[INFO] 💻 ✓ /dev successfully bound

======================
📊 Namespace Test Summary
======================
[SUMMARY]
[NAMESPACE UTS] 🌐 Hostname: namespace-lab
[NAMESPACE USER] 👤 UID/GID: uid=0(root) gid=0(root) groups=0(root)
[NAMESPACE PID] 🧵 Running processes:
        PID USER     COMMAND
          1 root     bash
          9 root     /usr/bin/ps -eo pid,user,args --sort=pid
         10 root     head -n 5
         11 root     sed s/^/    /
[NAMESPACE NET] 🌐 Network interfaces:
    1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
[NAMESPACE MOUNT] 📁/tmp write: ✓ OK
[NAMESPACE FS] 📁 Root filesystem:
bin dev etc lib lib64 proc sys tmp usr
======================


Press Enter to return to menu...
```

Understanding Cgroups (Control Groups)

 Light



Definition

Control Groups (cgroups) are a Linux kernel feature introduced in 2007 that allow you to limit, account for, and isolate the resource usage (CPU, memory, disk I/O, etc.) of groups of processes.

cgroups are heavily used by low-level container runtimes such as runc and crun, and leveraged by container engines like Docker, Podman, and LXC to enforce resource boundaries and provide isolation between containers.

Namespaces isolate, cgroups control.

Namespaces create separate environments for processes (like PID, network, or mounts), while cgroups limit and monitor resource usage (CPU, memory, I/O) for those processes.

 Key Capabilities

Feature	Description
Resource Limiting	Impose limits on how much of a resource a group can use
Prioritization	Allocate more CPU/IO priority to some groups over others

Light

Feature	Description
Accounting	Track usage of resources per group
Control	Suspend, resume, or kill processes in bulk
Isolation	Prevent resource starvation between groups

Subsystems (Controllers)

cgroups operate through controllers, each responsible for managing one type of resource:

Subsystem	Description
`cpu`	Controls CPU scheduling
`cpuacct`	Generates CPU usage reports
`memory`	Limits and accounts memory usage
`blkio`	Limits block device I/O
`devices`	Controls access to devices
`freezer`	Suspends/resumes execution of tasks
`net_cls`	Tags packets for traffic shaping
`ns`	Manages namespace access (rare)

Filesystem Layout

cgroups are exposed through the virtual filesystem under /sys/fs/cgroup.

Depending on the version:

- **cgroups v1**: separate hierarchies for each controller (e.g., memory, cpu, etc.)
- **cgroups v2**: unified hierarchy under a single mount point

Mounted under:

```
/sys/fs/cgroup/
```

Typical cgroups v1 hierarchy:

```
/sys/fs/cgroup/
├── memory/
│   ├── mygroup/
│   │   ├── tasks
│   │   ├── memory.limit_in_bytes
├── cpu/
```

Light

```
│   └── mygroup/
└── ...
```

In cgroups v2, all resources are managed under a unified hierarchy:

```
/sys/fs/cgroup/
├── cgroup.procs
├── cgroup.controllers
├── memory.max
├── cpu.max
└── ...
```

Common Usage (v1 and v2 examples)

v1 – Create and assign memory limit:

```
# Mount memory controller (if needed)
mount -t cgroup -o memory none /sys/fs/cgroup/memory

# Create group
mkdir /sys/fs/cgroup/memory/mygroup

# Set memory limit (100 MB)
echo 104857600 | tee /sys/fs/cgroup/memory/mygroup/memory.limit_in_bytes

# Assign a process (e.g., current shell)
echo $$ | tee /sys/fs/cgroup/memory/mygroup/tasks
```

v2 – Unified hierarchy:

```
# Create subgroup
mkdir /sys/fs/cgroup/mygroup

# Enable controllers
echo +memory +cpu > /sys/fs/cgroup/cgroup.subtree_control

# Move shell into group
echo $$ > /sys/fs/cgroup/mygroup/cgroup.procs

# Set limits
echo 104857600 > /sys/fs/cgroup/mygroup/memory.max
echo "50000 100000" > /sys/fs/cgroup/mygroup/cpu.max  # 50ms quota per 100ms period
```

Process & Group Inspection

Command	Description
`cat /proc/self/cgroup`	Shows current cgroup membership

Command	Description
`cat /proc/PID/cgroup`	cgroup of another process
`cat /proc/PID/status`	Memory and cgroup info
`ps -o pid,cmd,cgroup`	Show process-to-cgroup mapping

Usage in Containers

Container engines like Docker, Podman, and containerd delegate resource control to cgroups (via runc or crun), allowing:

- Per-container CPU and memory limits
- Fine-grained control over blkio and devices
- Real-time resource accounting

Docker example:

```
docker run --memory=256m --cpus=1 busybox
```

Behind the scenes, this creates cgroup rules for memory and CPU limits for the container process.

Concepts Summary

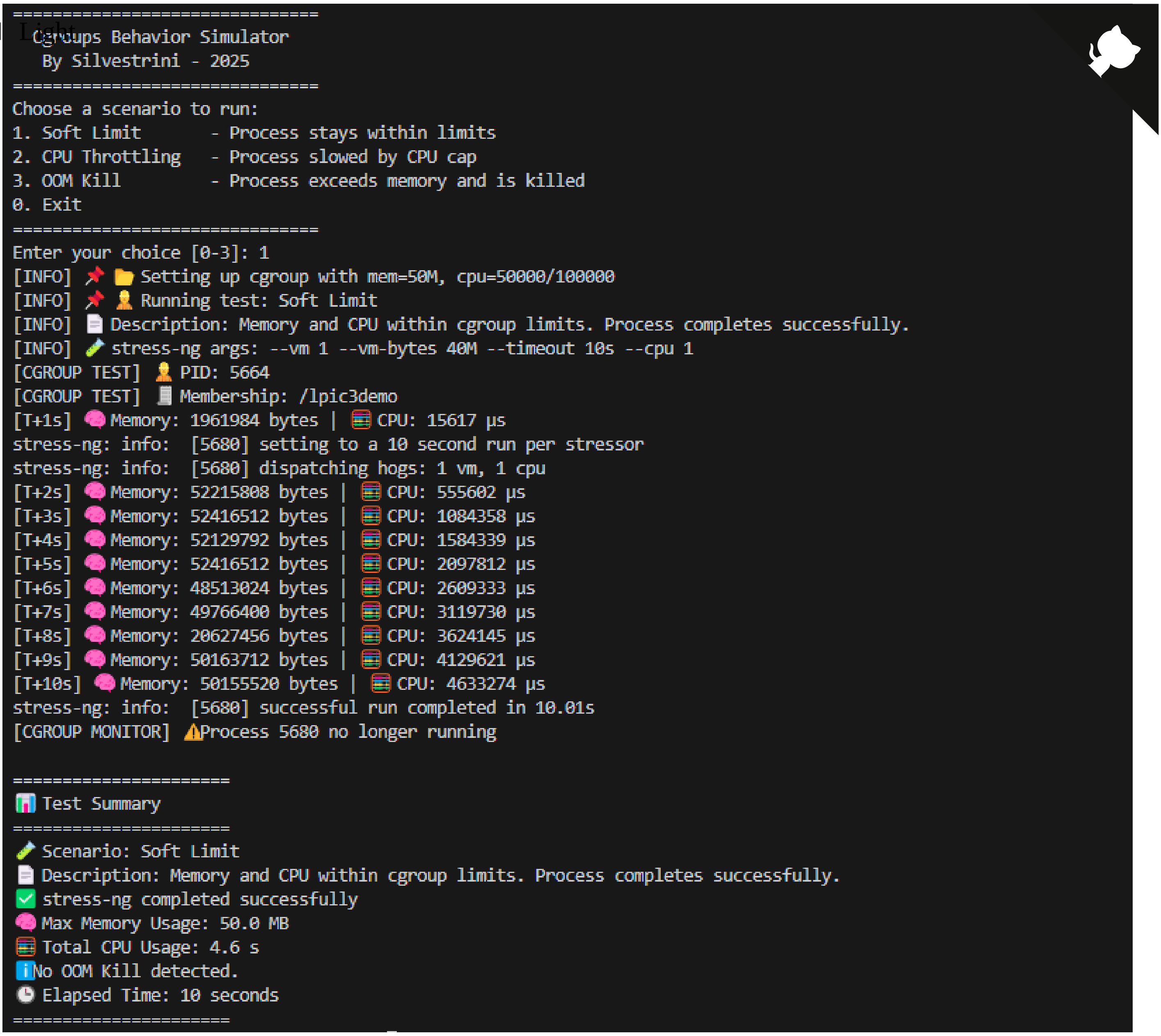
Concept	Explanation
Controllers	Modules like `cpu`, `memory`, `blkio`, etc. apply limits and rules
Tasks	PIDs (processes) assigned to the control group
Hierarchy	Cgroups are structured in a parent-child tree
Delegation	Systemd and user services may manage subtrees of cgroups

Lab Cgroups

Use this script for lab: cgroups.sh

Output Soft limit memory:

Light



```
==============================
  Cgroups Behavior Simulator
   By Silvestrini - 2025
==============================
Choose a scenario to run:
1. Soft Limit       - Process stays within limits
2. CPU Throttling   - Process slowed by CPU cap
3. OOM Kill         - Process exceeds memory and is killed
0. Exit
==============================
Enter your choice [0-3]: 1
[INFO] 📌 📁 Setting up cgroup with mem=50M, cpu=50000/100000
[INFO] 📌 👤 Running test: Soft Limit
[INFO] 📄 Description: Memory and CPU within cgroup limits. Process completes successfully.
[INFO] 🧪 stress-ng args: --vm 1 --vm-bytes 40M --timeout 10s --cpu 1
[CGROUP TEST] 👤 PID: 5664
[CGROUP TEST] 🗄 Membership: /lpic3demo
[T+1s] 🧠 Memory: 1961984 bytes | 🧮 CPU: 15617 µs
stress-ng: info:  [5680] setting to a 10 second run per stressor
stress-ng: info:  [5680] dispatching hogs: 1 vm, 1 cpu
[T+2s] 🧠 Memory: 52215808 bytes | 🧮 CPU: 555602 µs
[T+3s] 🧠 Memory: 52416512 bytes | 🧮 CPU: 1084358 µs
[T+4s] 🧠 Memory: 52129792 bytes | 🧮 CPU: 1584339 µs
[T+5s] 🧠 Memory: 52416512 bytes | 🧮 CPU: 2097812 µs
[T+6s] 🧠 Memory: 48513024 bytes | 🧮 CPU: 2609333 µs
[T+7s] 🧠 Memory: 49766400 bytes | 🧮 CPU: 3119730 µs
[T+8s] 🧠 Memory: 20627456 bytes | 🧮 CPU: 3624145 µs
[T+9s] 🧠 Memory: 50163712 bytes | 🧮 CPU: 4129621 µs
[T+10s] 🧠 Memory: 50155520 bytes | 🧮 CPU: 4633274 µs
stress-ng: info:  [5680] successful run completed in 10.01s
[CGROUP MONITOR] ⚠️Process 5680 no longer running

======================
📊 Test Summary
======================
🧪 Scenario: Soft Limit
📄 Description: Memory and CPU within cgroup limits. Process completes successfully.
✅ stress-ng completed successfully
🧠 Max Memory Usage: 50.0 MB
🧮 Total CPU Usage: 4.6 s
ℹ️No OOM Kill detected.
🕒 Elapsed Time: 10 seconds
======================
```

Understanding Capabilities

What Are Linux Capabilities?

Traditionally in Linux, the root user has unrestricted access to the system. Linux capabilities were introduced to break down these all-powerful privileges into smaller, discrete permissions, allowing processes to perform specific privileged operations without requiring full root access.

This enhances system security by enforcing the principle of least privilege.

Capability	**Description**
`CAP_CHOWN`	Change file owner regardless of permissions
`CAP_NET_BIND_SERVICE`	Bind to ports below 1024 (e.g., 80, 443)
`CAP_SYS_TIME`	Set system clock

Light **Capability**	**Description**
`CAP_SYS_ADMIN`	Very powerful – includes mount, BPF, and more
`CAP_NET_RAW`	Use raw sockets (e.g., ping, traceroute)
`CAP_SYS_PTRACE`	Trace other processes (debugging)
`CAP_KILL`	Send signals to any process
`CAP_DAC_OVERRIDE`	Modify files and directories without permission
`CAP_SETUID`	Change user ID (UID) of the process
`CAP_NET_ADMIN`	Manage network interfaces, routing, etc.

Some Linux Capabilities Types

Capability Type	Description
CapInh (Inherited)	Capabilities inherited from the parent process.
CapPrm (Permitted)	Capabilities that the process is allowed to have.
CapEff (Effective)	Capabilities that the process is currently using.
CapBnd (Bounding)	Restricts the maximum set of effective capabilities a process can obtain.
CapAmb (Ambient)	Allows a process to explicitly define its own effective capabilities.

Capabilities in Containers and Pods Containers typically do not run as full root, but instead receive a limited set of capabilities by default depending on the runtime.

Capabilities can be added or dropped in Kubernetes using the securityContext.

Kubernetes example:

```
securityContext:
  capabilities:
    drop: ["ALL"]
    add: ["NET_BIND_SERVICE"]
```

This ensures the container starts with zero privileges and receives only what is needed.

Lab Capabilities

Use this script for lab: capabilities.sh

Output:

Light

```
===============================
  Capabilities Simulator Lab
   By Silvestrini - 2025
===============================
Choose a scenario to run:
1. Drop All         - Packet capture fails (no CAP_NET_RAW)
2. Add CAP_NET_RAW  - Packet capture works
3. Bind Port 80     - Python binds to 80 without root
4. CAP_KILL         - Kill process with and without CAP_KILL
5. CAP_SYS_PTRACE   - Use strace on a process
6. CAP_SYS_ADMIN    - Try mounting tmpfs
9. Teardown/Cleanup Temporary Files
0. Exit
===============================
Enter your choice [0-9]: 2

======================
📊 Capability Test Summary
======================
🧪 Scenario: Add CAP_NET_RAW
📄 Description: Add only CAP_NET_RAW and attempt packet capture via tcpdump
🗄 Capability Status: Added CAP_NET_RAW
📈 Result: ✅ tcpdump ran – CAP_NET_RAW effective
======================

Press Enter to return to menu...
```

352.1 Important Commands

unshare

```
# create a new namespaces and run a command in it
unshare --mount --uts --ipc --user --pid --net  --map-root-user --mount-proc --fork
# mount /proc for test
#mount -t proc proc /proc
#ps -aux
#ip addr show
#umount /proc
```

lsns

```
# show all namespaces
lsns

# show only pid namespace
lsns -s <pid>
lsns -p 3669

ls -l /proc/<pid>/ns
ls -l /proc/3669/ns
```

```
ps -o pid,pidns,netns,ipcns,utsns,userns,args -p <PID>
ps -o pid,pidns,netns,ipcns,utsns,userns,args -p 3669
```

nsenter

```
# execute a command in namespace
sudo nsenter -t <PID> -n  ip link show
sudo nsenter -t 3669 -n ip link show
```

252.1 ip

```
# create a new network namespace
sudo ip netns add lxc1

# list network list
ip netns list

# exec command in network namespace
sudo ip netns exec lxc1 ip addr show
```

stat

```
# get cgroup version
stat -fc %T /sys/fs/cgroup
```

systemctl and systemd

```
# get cgroups of system
systemctl status
systemd-cgls
```

cgcreate

```
cgcreate -g memory,cpu:lsf
```

cgclassify

```
cgclassify -g memory,cpu:lsf <PID>
```

setcap cap_net_raw=ep /usr/bin/tcpdump

getcap /usr/bin/tcpdump

 Light

capsh - capability shell wrapper

(back to sub topic 352.1)

(back to topic 352)

(back to top)

352.2 LXC

Weight: 6

Description: Candidates should be able to use system containers using LXC and LXD. The version of LXC covered is 3.0 or higher.

Key Knowledge Areas:

- Understand the architecture of LXC and LXD
- Manage LXC containers based on existing images using LXD, including networking and storage
- Configure LXC container properties
- Limit LXC container resource usage
- Use LXD profiles
- Understand LXC images
- Awareness of traditional LXC tools

352.2 Cited Objects

```
lxd
lxc (including relevant subcommands)
```

352.2 Important Commands

foo

```
foo
```

(back to sub topic 352.2)

(back to top)

(back to top)

352.3 Docker

Weight: 9

Description: Candidate should be able to manage Docker nodes and Docker containers. This include understand the architecture of Docker as well as understanding how Docker interacts with the node's Linux system.

Key Knowledge Areas:

- Understand the architecture and components of Docker
- Manage Docker containers by using images from a Docker registry
- Understand and manage images and volumes for Docker containers
- Understand and manage logging for Docker containers
- Understand and manage networking for Docker
- Use Dockerfiles to create container images
- Run a Docker registry using the registry Docker image

352.3 Cited Objects

```
dockerd
/etc/docker/daemon.json
/var/lib/docker/
docker
Dockerfile
```

352.3 Important Commands

docker

```
# Examples of docker
```

(back to sub topic 352.3)

(back to topic 352)

(back to top)

Light

352.4 Container Orchestration Platforms

Weight: 3

Description: Candidates should understand the importance of container orchestration and the key concepts Docker Swarm and Kubernetes provide to implement container orchestration.

Key Knowledge Areas:

- Understand the relevance of container orchestration
- Understand the key concepts of Docker Compose and Docker Swarm
- Understand the key concepts of Kubernetes and Helm
- Awareness of OpenShift, Rancher and Mesosphere DC/OS

(back to sub topic 352.4)

(back to topic 352)

(back to top)

Topic 353: VM Deployment and Provisioning

353.1 Cloud Management Tools

Weight: 2

Description: Candidates should understand common offerings in public clouds and have basic feature knowledge of commonly available cloud management tools.

Key Knowledge Areas:

- Understand common offerings in public clouds
- Basic feature knowledge of OpenStack
- Basic feature knowledge of Terraform
- Awareness of CloudStack, Eucalyptus and OpenNebula

353.1 Cited Objects

```
IaaS, PaaS, SaaS
OpenStack
Terraform
```

353.1 Important Commands

```
# examples
```

(back to sub topic 353.1)

(back to topic 353)

(back to top)

353.2 Packer

Weight: 2

Description: Candidates should be able to use Packer to create system images. This includes running Packer in various public and private cloud environments as well as building container images for LXC/LXD.

Key Knowledge Areas:

- Understand the functionality and features of Packer
- Create and maintain template files
- Build images from template files using different builders

353.2 Cited Objects

```
packer
```

353.2 Important Commands

packer

```
# examples
```

(back to sub topic 353.2)

(back to topic 353)

(back to top)

353.3 cloud-init

Weight: 3

Description: Candidates should able to use cloud-init to configure virtual machines create standardized images. This includes adjusting virtual machines to match their available hardwa resources, specifically, disk space and volumes.
Additionally, candidates should be able to configure instances to allow secure SSH logins and install a specific set of software packages.
Furthermore, candidates should be able to create new system images with cloud-init support.

Key Knowledge Areas:

- Understanding the features and concepts of cloud-init, including user-data, initializing and configuring cloud-init
- Use cloud-init to create, resize and mount file systems, configure user accounts, including login credentials such as SSH keys and install software packages from the distribution's repository
- Integrate cloud-init into system images
- Use config drive datasource for testing

353.3 Cited Objects

```
cloud-init
user-data
/var/lib/cloud/
```

353.3 Important Commands

foo

```
# examples
```

(back to sub topic 353.3)

(back to topic 353)

(back to top)

353.4 Vagrant

Weight: 3

Description: Candidate should be able to use Vagrant to manage virtual machines, including provisioning of the virtual machine.

Key Knowledge Areas:

 Light

- Understand Vagrant architecture and concepts, including storage and networking
- Retrieve and use boxes from Atlas
- Create and run Vagrantfiles
- Access Vagrant virtual machines
- Share and synchronize folder between a Vagrant virtual machine and the host system
- Understand Vagrant provisioning, i.e. File and Shell provisioners
- Understand multi-machine setup

353.4 Cited Objects

```
vagrant
Vagrantfile
```

353.4 Important Commands

vagrant

```
# examples
```

(back to sub topic 353.4)

(back to topic 353)

(back to top)

Contributing

Contributions are what make the open source community such an amazing place to learn, inspire, and create. Any contributions you make are **greatly appreciated**.

If you have a suggestion that would make this better, please fork the repo and create a pull request. You can also simply open an issue with the tag "enhancement". Don't forget to give the project a star! Thanks again!

1. Fork the Project
2. Create your Feature Branch (`git checkout -b feature/AmazingFeature`)
3. Commit your Changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the Branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

License

Contact

Marcos Silvestrini - marcos.silvestrini@gmail.com
Follow @mrsilvestrini

Project Link: https://github.com/marcossilvestrini/learning-lpic-3-305-300

(back to top)

Acknowledgments

- Richard Stallman's
- GNU
  - GNU/Linux FAQ by Richard Stallman
  - GNU
  - GNU Operating System
  - GCC Compiler
  - GNU Tar
  - GNU Make
  - GNU Emacs
  - GNU Packages
  - GNU/Linux Collection
  - GNU Grub Bootloader
  - GNU Hurd
- Kernel
  - Kernel
  - Linux Kernel Man Pages
  - Compile Your Kernel
- Linux Standard Base
  - Linux Standard Base
  - Filesystem Hierarchy Standard
  - File Hierarchy Structure
- Free Software
  - FSF

Light

    - Free Software Directory
- License
    - Free Software
    - Copyleft
    - GPL
    - GNU Lesser General Public License
    - BSD
    - Open Source Initiative
    - Creative Commons
    - License LTS
- Distros
    - Debian Free Software Guidelines
    - List Linux Distribution
    - Distro Watch
    - Comparison Linux Distributions
- Desktop Environments
    - X11 Org
    - Wayland
    - GNU GNOME
    - GNOME
    - XFCE
    - KDE Plasma
    - Harmony
- Protocols
    - HTTP
        - W3Techs
        - Apache
        - Apache Directives
        - HTTP Status Codes
        - Strong Ciphers for Apache, nginx and Lighttpd
        - SSL Tutorials
        - SSL Config Mozilla
    - xRDP
    - NTP
- DNS
    - Bind
    - Bind Logging
    - List of DNS record types

Light
  - List of DNS record types
- Package Manager
  - Download Packages
  - Install Packages
  - Guide Install Packages
- Shell Script
  - Bourne Again Shell
  - Shebang
  - Environment Variables
  - GNU Globbing
  - Globbing
  - Quoting
  - Regular Expressions
  - Command Not Found
  - Bash Prompt Generator
  - Explainshell
  - Vim Tutorial
  - Linux Shell Scripting Tutorial
  - Commands Examples
- Others Tools
  - Bugzila
  - Github Badges
- Virtualization Definitions
  - Red Hat
  - AWS
  - IBM
  - OpenSource.com
- Xen
  - XenServer
  - Wiki XenProject
  - Network Interfaces
  - Xen Tools
  - LPI Blog: Xen Virtualization and Cloud Computing #01: Introduction
  - LPI Blog: Xen Virtualization and Cloud Computing #02: How Xen Does the Job
  - LPI Blog: Xen Virtualization and Cloud Computing #04: Containers, OpenStack, and Other Related Platforms
  - Xen Virtualization and Cloud Computing #05: The Xen Project, Unikernels, and the Future

- Xen Project Beginners Guide
- Crazy Book

- Unikernels
  - Unikraft
  - MirageOS
  - HaLVM
  - Unik
- KVM
  - Oficial Doc
  - KVM(Kernel Virtual Machines by RedHat)
  - KVM Management Tools
  - KVM Network
- QEMU
  - Oficial Doc
  - Download Images osboxes
  - Download Images linuximages
  - Virtio
  - Guest Agent
- Libvirt
  - Oficial Doc
  - System Socket Activation
  - Conections
  - Storage
  - Network
  - VirtualNetwork
  - virtlogd
  - virtlockd
  - virt-manager
- Disk Managment
  - Disk Images
  - copy-on-write
  - RAM x QCOW2
  - Libguestfs
- Virtualization and Containerization
  - AWS Doc Containers
  - GCP Doc Containers
  - IBM Doc Container

 Light

- Red Hat Docs Containers
- Namespaces
- Most important Namespaces
- Cgroups Classes
- Man Cgroups
- Capabilities Doc
- Man Capabilities
-

- Openstack Docs
  - RedHat
- Open vSwitch
  - OVS Doc 4Linux
- LPIC-3 305-300 Exam
  - LPIC-3 305-300 Objectives
  - LPIC-3 305-300 Wiki
  - LPIC-3 305-300 Learning Material
  - LPIC-3 305-300 Simulated Exam By ITexams

(back to top)