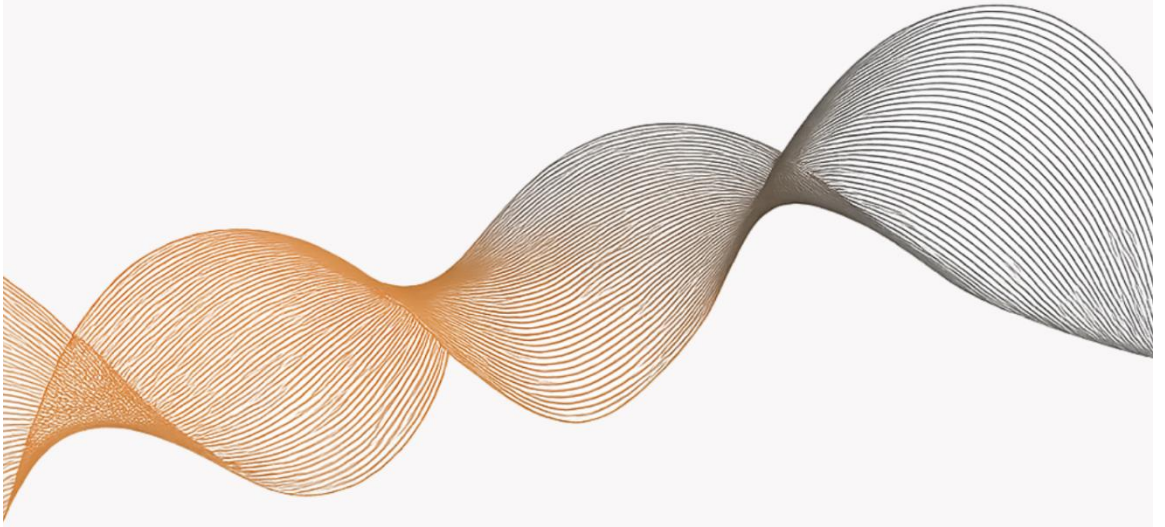
ENTERPRISE VIRTUALIZATION

# MASTERING PROXMOX VE 9

*Building Enterprise Clusters with Ceph Storage*



**Enterprise Edition – Version 9.0**

**Bekroundjo Akoley Aristide**

# Copyright Page



For permission requests, write to:
**EC INTELLIGENCE**
La Marina Casablanca, Tour Océanes 3 Bureau 03 Rez-de-Jardin
Casablanca, Maroc
Website: https://ecintelligence.ma

**ISBN (Paperback):** 9798299298017

Printed in the United States of America

# Preface

## Purpose of the Book

The goal of *Mastering Proxmox VE 9: Building Enterprise Clusters with Ceph Storage* is to guide IT professionals through the deployment, configuration, and optimization of Proxmox VE in production-ready environments. While Proxmox is often associated with small or mid-sized labs, this book demonstrates how it can scale to support enterprise-grade infrastructures by integrating advanced clustering and Ceph storage features.

This book is designed to bridge the gap between initial installation and real-world, highly available deployments. It provides both the conceptual understanding and practical tools required to build resilient clusters with confidence.

## Audience

This book is written for:

- **System Administrators** managing virtualization environments.
- **Architects** designing scalable and fault-tolerant clusters.
- **DevOps Engineers** who need automation and integration with modern toolchains.
- **Students and Enthusiasts** seeking to learn virtualization and distributed storage hands-on.

No prior Proxmox experience is strictly required, but readers should be familiar with Linux administration, networking concepts, and basic virtualization.

## What Readers Will Learn

By following the chapters, readers will gain knowledge and skills in:

- Deploying Proxmox VE 9 from scratch.
- Building clusters with multiple nodes.
- Integrating and managing Ceph storage.
- Configuring High Availability (HA) for virtual machines and containers.
- Using advanced networking, fencing, and replication.
- Monitoring and troubleshooting complex environments.
- Automating deployments with Ansible and CLI tools.

## How the Book Is Structured

The book is divided into ten core chapters, each building upon the previous one.

- Early chapters introduce **installation, networking, and cluster basics**.
- Midway, the focus shifts to **Ceph storage integration, scaling clusters, and HA strategies**.
- Later chapters cover **automation, performance optimization, and troubleshooting**.
- Supporting sections, such as the Appendix and Glossary, provide quick references and advanced tools.

# How to Use This Book

### Lab Setup Requirements

To follow the examples in this book, you will need:

- At least **three physical or virtual servers** with 16 GB RAM, multi-core CPUs, and local storage.
- A **10 GbE network** (recommended) for cluster and Ceph traffic.
- Proxmox VE 9 ISO image and access to the Proxmox repositories.
- Optional: SSDs or NVMe devices to simulate high-performance Ceph clusters.

### Style Conventions

- **Code blocks** are shown in monospaced text with grey background.
- **Important notes** highlight crucial details for production deployments.
- **Tables and diagrams** summarize configurations, commands, and architectures.
- Commands are written for **Debian-based Linux** (as Proxmox is based on Debian).

### Beginner vs. Advanced Content

- Beginner-friendly explanations are included in the first half of each chapter.
- Advanced topics, such as **NVMe-oF gateways, CRUSH map tuning, and fencing mechanisms**, are introduced later in the book.
- Readers may skip directly to advanced sections if they already have experience with Proxmox basics.

# Intellectual Property Notice

Proxmox is a trademark of **Proxmox Server Solutions GmbH**.
Ceph is a trademark of **Red Hat, Inc.**

This book is an **independent publication**. It has not been authorized, sponsored, or otherwise approved by Proxmox Server Solutions GmbH, Red Hat, Inc., or any of their affiliates.

All trademarks, logos, and brand names used in this book are the property of their respective owners. They are used in an editorial manner with the sole purpose of accurately describing the software platforms discussed.

The author and publisher make no claim of ownership over these trademarks.

# Acknowledgments

I would like to thank the many professionals and students who provided feedback and insights during the creation of this book. Special thanks to the team at **EC INTELLIGENCE** for their continued support in both technical validation and project delivery. This book is also dedicated to the broader open-source community, whose contributions make Proxmox and Ceph powerful technologies accessible to all.

# About the Author

**Akoley Aristide Bekroundjo** is a Cloud Architect, trainer, and founder of **EC INTELLIGENCE**, a company dedicated to helping enterprises build modern IT infrastructures. With more than a decade of hands-on experience, he has designed and deployed **Proxmox, Ceph, OpenStack, and OpenShift clusters** for organizations across Africa, Europe, and the Americas.

Aristide has also delivered professional training to hundreds of engineers, administrators, and architects, bridging the gap between open-source innovation and enterprise-class requirements. His mission is to make high-availability, scalable, and secure infrastructures accessible to companies of all sizes.

When he is not teaching or architecting complex environments, he shares his insights through books, articles, and technical videos, always aiming to translate advanced concepts into clear, actionable knowledge.

**Connect with the Author**
LinkedIn: linkedin.com/in/akoley-bekroundjo

# Table of Contents

# Chapter 1: Introduction to Proxmox VE 9

The landscape of IT infrastructure has undergone a fundamental transformation over the past decade. Organizations of all sizes are seeking virtualization solutions that provide enterprise-grade capabilities without the complexity and cost traditionally associated with such platforms. In this evolving ecosystem, Proxmox Virtual Environment (VE) has emerged as a compelling alternative, offering a unique combination of power, flexibility, and accessibility that challenges the established order of proprietary virtualization platforms.

This chapter provides a comprehensive introduction to Proxmox VE 9, exploring not just what it is, but why it represents a paradigm shift in how we approach virtualization. We'll examine the architectural decisions that make Proxmox VE both powerful and approachable, understand its position in the modern infrastructure stack, and explore the groundbreaking features introduced in version 9 that address long-standing enterprise requirements.

## 1.1 What is Proxmox VE?

To understand Proxmox VE, we must first recognize that it is not merely another hypervisor or virtualization tool. Rather, it represents a complete virtualization management platform that seamlessly integrates multiple virtualization technologies, storage systems, networking capabilities, and management tools into a cohesive whole. This integration is what sets Proxmox VE apart from solutions that require administrators to cobble together various components to achieve similar functionality.

### The Evolution of Virtualization Platforms

The journey of virtualization technology has been marked by several distinct phases. Initially, virtualization was the domain of mainframes, where resource partitioning was essential for multi-tenant operations. The x86 virtualization revolution, sparked by VMware in the late 1990s, brought these capabilities to commodity hardware. However, as virtualization became mainstream, two distinct challenges emerged:

1. **Complexity**: Enterprise virtualization platforms became increasingly complex, requiring specialized knowledge and significant resources to deploy and maintain.
2. **Cost**: Licensing models for commercial virtualization platforms often represented a significant portion of IT budgets, particularly for small to medium enterprises.

Proxmox VE emerged from the recognition that open-source technologies had matured to the point where they could provide enterprise-grade virtualization capabilities without these traditional barriers. By leveraging the best of open-source virtualization technologies and wrapping them in a unified management layer, Proxmox VE democratizes access to advanced virtualization features.

## Core Virtualization Technologies

At its heart, Proxmox VE integrates two complementary virtualization approaches, each suited to different use cases:

### KVM (Kernel-based Virtual Machine)

KVM represents the pinnacle of Type-1 hypervisor technology in the Linux ecosystem. Unlike Type-2 hypervisors that run as applications on top of a host operating system, KVM transforms the Linux kernel itself into a hypervisor. This architectural approach provides several critical advantages:

- **Performance**: By operating at the kernel level, KVM can achieve near-native performance for guest operating systems. The hypervisor overhead is minimal, typically less than 3% for most workloads.
- **Hardware Integration**: KVM leverages hardware virtualization extensions (Intel VT-x and AMD-V) to provide hardware-assisted virtualization. This includes support for nested page tables (EPT/NPT), which significantly reduces the overhead of memory management in virtualized environments.
- **Security Isolation**: Each KVM virtual machine runs as a separate Linux process, benefiting from the robust security model of the Linux kernel. This includes SELinux/AppArmor integration, memory protection, and process isolation.
- **Device Support**: Through QEMU (Quick Emulator), KVM can emulate a wide range of hardware devices, from legacy ISA devices to modern PCIe controllers. This flexibility allows running virtually any x86-based operating system.

### LXC (Linux Containers)

While KVM provides full system virtualization, LXC offers a different approach through OS-level virtualization. This technology, which shares conceptual similarities with Docker but predates it, provides isolated Linux environments without the overhead of running separate kernels:

- **Resource Efficiency**: LXC containers share the host kernel, eliminating the memory and CPU overhead associated with running multiple kernel instances. A typical LXC container might consume as little as 10MB of RAM for a minimal Linux environment.
- **Instant Startup**: Without the need to boot a separate kernel, LXC containers can start in milliseconds rather than the seconds or minutes required for traditional VMs.
- **Native Performance**: Since containers run on the host kernel, there's no virtualization overhead for system calls or hardware access. This makes them ideal for I/O-intensive applications.
- **Density**: The reduced overhead allows running hundreds or even thousands of containers on a single host, far exceeding what's practical with full virtualization.

## The Proxmox VE Integration Layer

What transforms these individual technologies into a comprehensive platform is the Proxmox VE integration layer. This layer consists of several key components:

### Web-based Management Interface

The Proxmox VE web interface represents more than just a GUI; it's a complete management console that provides:

- **Multi-node Management**: A single interface can manage an entire cluster, providing a unified view of all resources regardless of their physical location.
- **Real-time Monitoring**: Live statistics for CPU, memory, network, and storage usage across all nodes and guests, with historical data for trend analysis.
- **Integrated Console Access**: Both noVNC and SPICE protocols are supported for accessing guest consoles directly through the web browser, eliminating the need for separate client software.
- **Task Management**: All operations are executed as background tasks with full logging, allowing administrators to track the status and history of all actions.

### RESTful API

Every feature available in the web interface is backed by a comprehensive RESTful API. This design philosophy ensures:

- **Automation First**: Any task that can be performed through the GUI can be automated via the API, enabling infrastructure-as-code approaches.
- **Language Agnostic**: The REST API can be consumed by any programming language or tool that supports HTTP, from simple shell scripts using curl to sophisticated orchestration platforms.
- **Complete Documentation**: The API is self-documenting, with schema definitions and examples available for every endpoint.

### Proxmox Cluster File System (pmxcfs)

One of the most innovative aspects of Proxmox VE is its cluster file system. Unlike traditional shared storage approaches, pmxcfs provides:

- **Configuration Synchronization**: All cluster configuration is automatically replicated across nodes in real-time using Corosync's reliable multicast protocol.
- **Version Control**: Built-in versioning allows tracking configuration changes and rolling back if needed.
- **High Performance**: The entire configuration database is kept in RAM on each node, providing microsecond access times for configuration queries.

- **Fault Tolerance**: The distributed nature ensures that configuration remains available even if multiple nodes fail, as long as quorum is maintained.

## Understanding the Architecture Through Comparison

To fully appreciate Proxmox VE's architectural approach, it's instructive to compare it with other virtualization models. Consider the following architectural diagram that illustrates how Proxmox VE differs from traditional virtualization stacks:

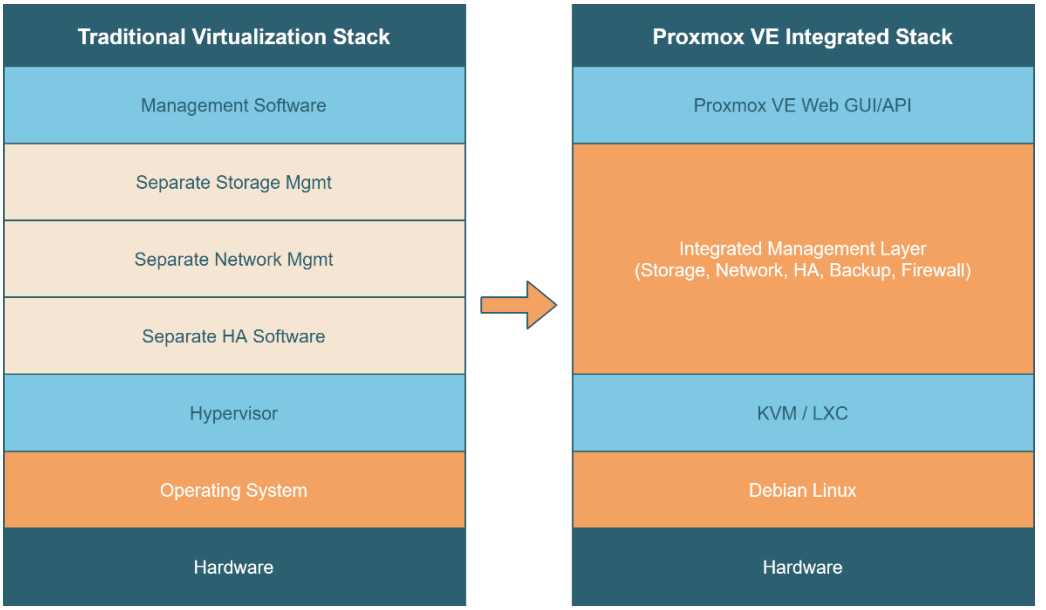



*Figure 1 Architecture Stack Comparison*

This integrated approach eliminates the complexity of managing multiple software components while providing a more cohesive and efficient system.

# 1.2 Why Choose Proxmox VE for Virtualization

The decision to adopt a virtualization platform extends far beyond technical specifications. It encompasses considerations of cost, support, ecosystem, and long-term viability. Proxmox VE addresses each of these concerns in ways that challenge traditional assumptions about enterprise virtualization.

## Economic Considerations

The economic argument for Proxmox VE is compelling, but it extends beyond the obvious benefit of free software. Consider the total cost of ownership (TCO) for a typical virtualization deployment:

**Traditional Commercial Platform Costs:**

- Base hypervisor licensing: $3,000-$5,000 per CPU socket
- Management software: $5,000-$50,000 depending on scale
- Advanced features (vMotion, HA, replication): Additional per-VM or per-CPU costs
- Annual support and maintenance: 20-25% of license costs
- Required training and certification: $3,000-$5,000 per administrator

**Proxmox VE Cost Structure:**

- Software licenses: $0
- Optional support subscription: €90-€874 per year per physical CPU
- Training: Community resources available at no cost
- Advanced features: All included in base installation

For a typical small to medium enterprise with 3 hosts, each with 2 CPU sockets, the first-year cost difference can exceed $50,000, with similar savings recurring annually.

## Feature Parity and Beyond

One might assume that a free, open-source platform would lag behind commercial offerings in features. However, Proxmox VE not only matches but in many cases exceeds the capabilities of proprietary platforms:

**Storage Flexibility**

While commercial platforms often require specific storage hardware or additional licensing for advanced storage features, Proxmox VE provides native support for:

- **Local Storage**: LVM, LVM-thin, directory-based, ZFS
- **Network Storage**: NFS, CIFS/SMB, iSCSI (both kernel and user-mode), GlusterFS
- **Distributed Storage**: Ceph RBD and CephFS with full integration
- **Advanced Features**: Thin provisioning, snapshots, clones, and live storage migration included at no additional cost

**Networking Capabilities**

The networking stack in Proxmox VE leverages the full power of Linux networking, providing:

- **Software-Defined Networking**: Native integration with Open vSwitch for advanced switching features
- **VLAN Support**: Full 802.1Q VLAN tagging and trunking
- **Bonding**: Multiple bonding modes for redundancy and performance
- **Advanced Routing**: Full support for static and dynamic routing protocols
- **Firewall**: Integrated datacenter-wide firewall with granular rule management

### High Availability Without Complexity

Unlike commercial solutions that require separate clustering software or management platforms, Proxmox VE includes:

- **Integrated HA Stack**: Based on proven technologies (Corosync, Pacemaker)
- **Automatic Failover**: VMs automatically restart on healthy nodes if a host fails
- **Fencing Support**: Multiple fencing methods to ensure data integrity
- **No Additional Licensing**: HA features available for unlimited VMs

## Open Standards and Vendor Independence

The commitment to open standards in Proxmox VE provides strategic advantages that extend beyond cost savings:

### No Vendor Lock-in

Every component of Proxmox VE uses standard, open formats:

- VM disks use standard qcow2, raw, or VMDK formats
- Configuration files are human-readable text
- Network configurations use standard Linux networking
- APIs follow REST principles with JSON data formats

This approach ensures that:

- Migration to or from other platforms remains possible
- Integration with third-party tools is straightforward
- Custom automation and tooling can be developed without proprietary SDKs

### Community and Ecosystem

The Proxmox VE ecosystem demonstrates the power of open-source collaboration:

- **Active Community**: Over 170,000 registered forum members contributing knowledge and solutions
- **Third-party Integration**: Extensive ecosystem of backup solutions, monitoring tools, and automation platforms
- **Transparent Development**: Public bug tracker and roadmap ensure visibility into future development
- **Regular Updates**: Predictable release cycle with long-term support versions

## Performance Characteristics

Performance is often cited as a concern when considering open-source solutions. However, extensive benchmarking reveals that Proxmox VE often outperforms commercial alternatives:

### Virtualization Overhead

Independent benchmarks consistently show:

- KVM CPU overhead: <2% for most workloads
- Memory overhead: 300-500MB per VM for KVM, <50MB for LXC
- I/O performance: Within 5% of bare metal with proper configuration
- Network performance: Line-rate performance with SR-IOV or virtio

### Scalability Metrics

Real-world deployments demonstrate impressive scalability:

- Clusters with 30+ nodes in production
- Single nodes running 200+ VMs or 1000+ containers
- Storage pools exceeding 1PB
- Networks handling 100Gbps+ aggregate throughput

## Security Architecture

Security in Proxmox VE is built on the proven foundation of Linux security models, enhanced with virtualization-specific protections:

### Multi-layered Security Model

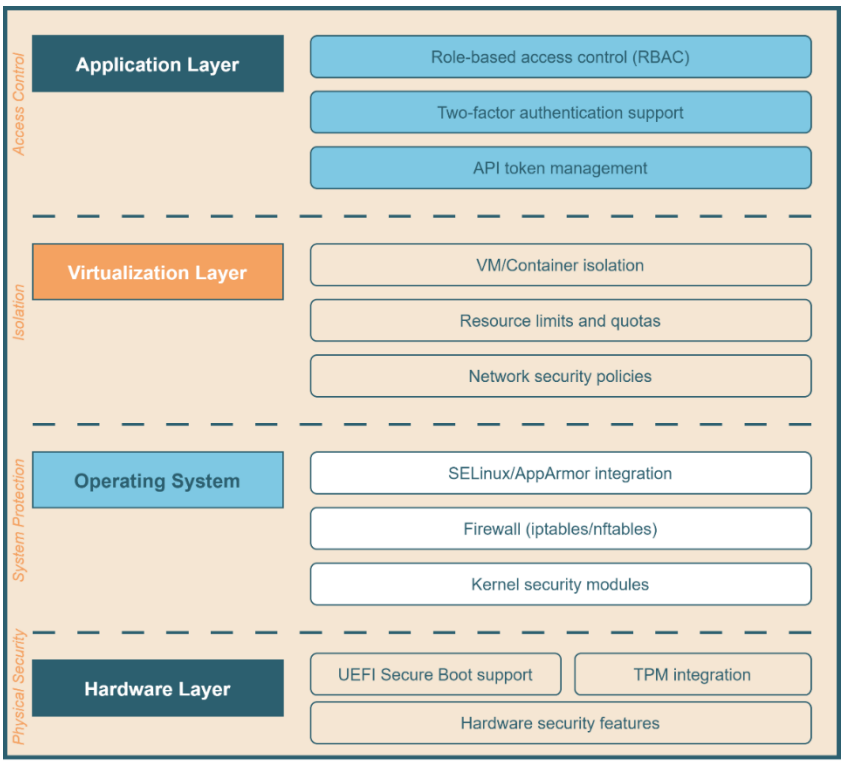



*Figure 2 Proxmox VE Security Architecture*

### Compliance and Auditing

For organizations with compliance requirements:

- Comprehensive audit logging of all actions
- Integration with external authentication systems (LDAP, AD)
- Support for encryption at rest and in transit
- Regular security updates through Debian security team

## 1.3 Understanding the Architecture

A deep understanding of Proxmox VE's architecture is essential for designing, deploying, and maintaining effective virtualization infrastructure. This section explores the technical underpinnings that make Proxmox VE both powerful and reliable.

### The Hypervisor Layer in Detail

The hypervisor layer in Proxmox VE represents a sophisticated integration of multiple technologies, each carefully chosen and configured for optimal performance and reliability.

### KVM Architecture Deep Dive

KVM's integration into the Linux kernel provides unique advantages that distinguish it from monolithic hypervisor designs. When a system boots Proxmox VE, the following initialization sequence occurs:

1. **Kernel Module Loading**: The kvm.ko module loads, along with processor-specific modules (kvm-intel.ko or kvm-amd.ko)
2. **Hardware Feature Detection**: The system queries CPU capabilities including:
    - Virtualization extensions (VMX/SVM)
    - Extended Page Tables (EPT/NPT)
    - Virtual Processor Identifiers (VPID)
    - Posted Interrupts support
3. **QEMU Process Architecture**: Each VM runs as a QEMU process with:
    - Dedicated memory allocation
    - CPU thread per vCPU
    - I/O threads for disk and network operations
    - Monitor thread for management operations

The relationship between these components can be visualized as:

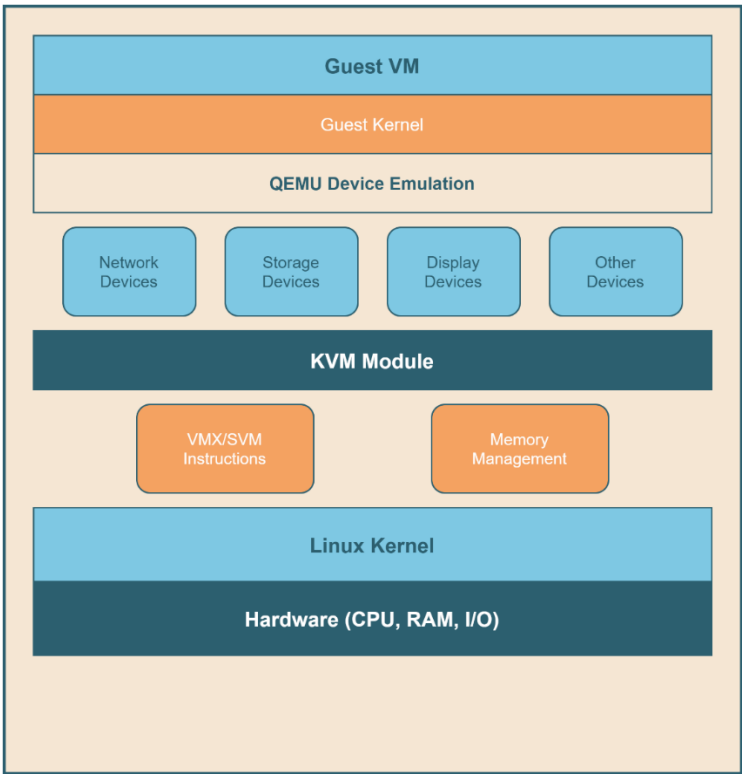



*Figure 3 KVM Guest VM Architecture*

### Memory Management in KVM

One of the most critical aspects of virtualization performance is memory management. KVM employs several sophisticated techniques:

**Transparent Huge Pages (THP)**: By default, KVM attempts to use 2MB or 1GB huge pages instead of standard 4KB pages. This reduces:

- TLB (Translation Lookaside Buffer) pressure
- Page table walk overhead
- Memory management overhead

**Kernel Same-page Merging (KSM)**: For environments running multiple similar VMs, KSM can:

- Identify identical memory pages across VMs
- Merge them into a single copy-on-write page
- Potentially save 20-60% of memory in VDI environments

**Memory Ballooning**: The virtio-balloon driver allows dynamic memory management:

- Host can request VMs to release unused memory
- Memory can be redistributed to VMs under pressure

- Automatic balancing based on VM activity

**NUMA Optimization**: For multi-socket systems, KVM provides:

- NUMA-aware memory allocation
- vCPU pinning to specific NUMA nodes
- Optimal memory access patterns for large VMs

### LXC Architecture and Implementation

While KVM provides full system virtualization, LXC offers a fundamentally different approach that's particularly relevant for Linux workloads:

### Container Isolation Mechanisms

LXC leverages multiple Linux kernel features to provide isolation:

1. **Namespaces**: Provide isolated views of system resources
    - PID namespace: Separate process trees
    - Network namespace: Isolated network stacks
    - Mount namespace: Independent filesystem views
    - User namespace: UID/GID mapping
    - UTS namespace: Separate hostnames
    - IPC namespace: Isolated inter-process communication
2. **Control Groups (cgroups)**: Resource limitation and accounting
    - CPU shares and quotas
    - Memory limits and accounting
    - I/O bandwidth and IOPS limits
    - Device access control
3. **Security Modules**: Additional security layers
    - AppArmor profiles for application confinement
    - SELinux contexts for mandatory access control
    - Seccomp filters for system call filtering

The complete isolation model can be represented as:

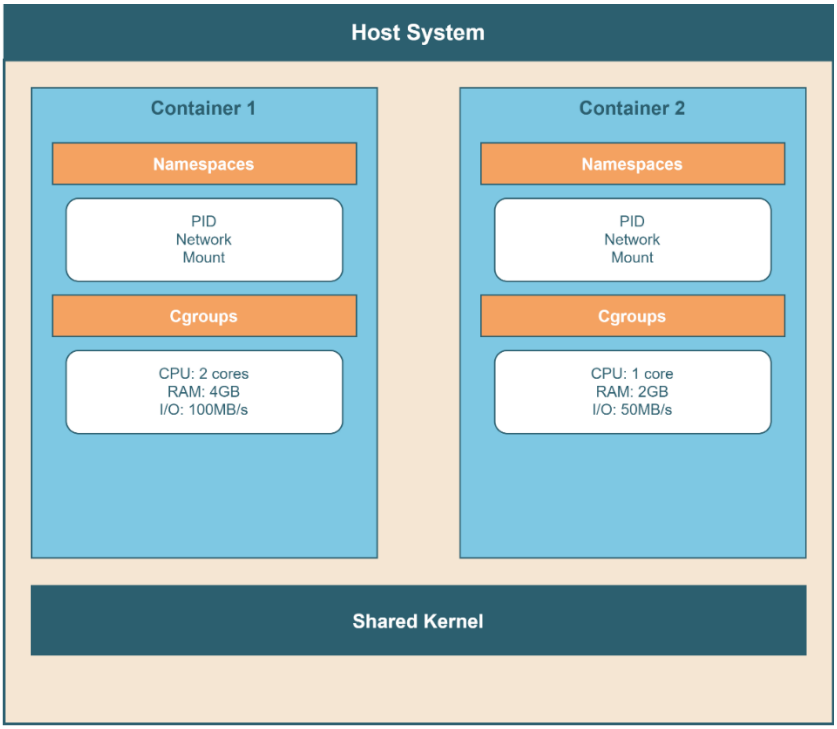



*Figure 4 LXC Container Isolation Model*

## Storage Architecture in Depth

The storage architecture in Proxmox VE represents one of its greatest strengths, providing flexibility that matches or exceeds commercial solutions while maintaining simplicity of management.

### Storage Abstraction Layer

The Proxmox VE storage model uses a plugin-based architecture that provides:

1. **Uniform Interface**: Regardless of the underlying storage type, all storage operations use the same API
2. **Storage Pools**: Logical grouping of storage resources
3. **Content Types**: Differentiation between VM images, containers, ISOs, templates, and backups
4. **Allocation Tracking**: Automatic tracking of disk usage and ownership

### Storage Plugin Architecture

Each storage type is implemented as a plugin that must provide specific operations:

```
# Simplified plugin interface
sub activate_storage {
```

```
    # Make storage available for use
}

sub deactivate_storage {
    # Cleanly disconnect storage
}

sub alloc_image {
    # Allocate new disk image
    return $volid;
}

sub free_image {
    # Delete disk image
}

sub clone_image {
    # Create linked clone if supported
}

sub create_snapshot {
    # Create point-in-time snapshot
}
```

### File-based vs Block-based Storage

Understanding the distinction between file-based and block-based storage is crucial for optimal deployment:

**File-based Storage Characteristics:**

- Flexibility in file naming and organization
- Easy backup and migration via file copy
- Support for any POSIX-compliant filesystem
- Potential for fragmentation over time
- Examples: Directory, NFS, CIFS, GlusterFS

**Block-based Storage Characteristics:**

- Better performance for random I/O
- Native snapshot support in most implementations
- More efficient space utilization
- Direct integration with VM block devices
- Examples: LVM, iSCSI, Ceph RBD, ZFS zvols

### Advanced Storage Features

Several storage backends provide advanced features that enable enterprise use cases:

**Thin Provisioning:** Supported by LVM-thin, ZFS, and Ceph

- Allocate storage on-demand as data is written

- Overcommit physical storage safely
- Monitor usage to prevent exhaustion

**Snapshots and Clones:** Implementation varies by storage type

- **Copy-on-Write Snapshots:** ZFS, Ceph, LVM-thin
- **Linked Clones:** ZFS, Ceph with RBD
- **Qcow2 Internal Snapshots:** Directory, NFS, CIFS

**Live Storage Migration:** Move disks between storage pools without downtime

- Supported for most storage combinations
- Automatic format conversion if needed
- Bandwidth limiting to prevent network saturation

## Network Architecture and SDN

The networking capabilities in Proxmox VE 9 have evolved to meet the demands of modern software-defined infrastructure.

### Traditional Linux Bridge Networking

The simplest networking configuration uses Linux bridges:

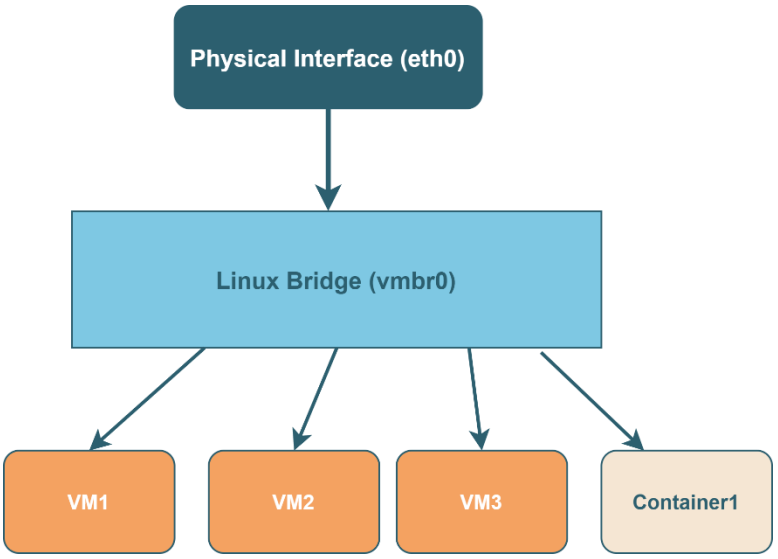



*Figure 5 Network bridge Topology*

This model provides:

- Simple configuration

- Good performance for basic scenarios
- VLAN support via bridge VLAN filtering
- Suitable for most small to medium deployments

### Open vSwitch Integration

For advanced networking requirements, Open vSwitch provides:

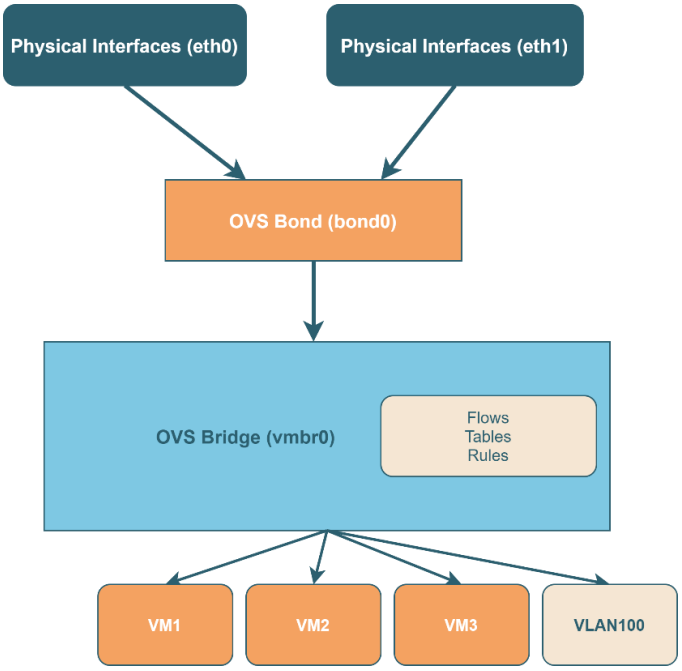



*Figure 6 OpenvSwitch Bond Topology*

Capabilities include:

- Flow-based forwarding
- Advanced VLAN handling
- Network virtualization overlays
- Integration with SDN controllers
- Traffic shaping and QoS

### Software-Defined Networking (SDN) in Proxmox VE 9

The new SDN framework introduces powerful abstractions:

**Zones**: Logical network segments

- Simple: Isolated layer 2 networks
- VLAN: Traditional VLAN-based segmentation

- QinQ: Provider bridging for service providers
- VXLAN: Overlay networks for multi-site deployments
- EVPN: BGP-based overlays for large-scale deployments

**VNets**: Virtual networks within zones

- Subnet management with IPAM
- DHCP integration
- DNS configuration
- Gateway management

**SDN Fabrics**: New in version 9

- Automated underlay configuration
- Dynamic routing protocol support
- Multi-path networking
- Simplified spine-leaf deployments

## Cluster Architecture and Distributed Systems

The cluster architecture in Proxmox VE represents a masterclass in distributed systems design, providing high availability without the complexity typically associated with such systems.

### Corosync Cluster Stack

At the heart of Proxmox VE clustering lies Corosync, which provides:

1. **Reliable Group Communication**: Ensures all nodes have consistent cluster state
2. **Membership Management**: Tracks which nodes are active and reachable
3. **Quorum Calculation**: Prevents split-brain scenarios
4. **Virtual Synchrony**: Guarantees message ordering across all nodes

### Cluster Communication Patterns

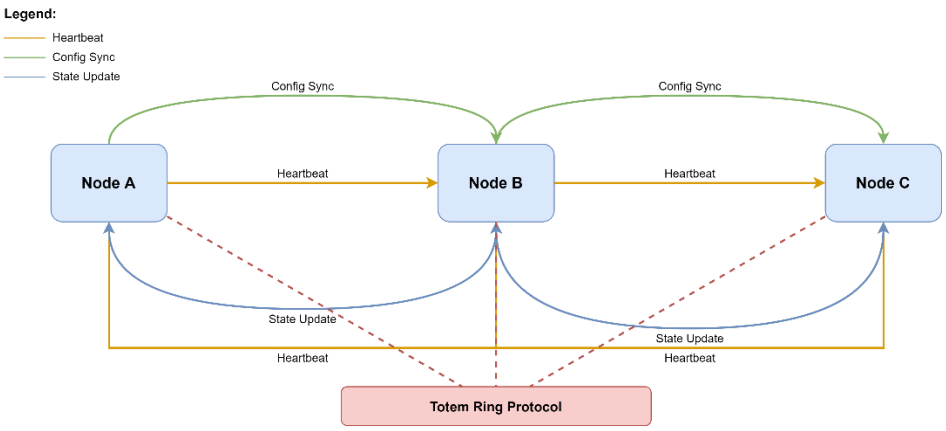



*Figure 7 Cluster communication Pattern*

### Distributed Configuration Management

The Proxmox Cluster File System (pmxcfs) provides:

1. **Real-time Replication**: Configuration changes propagate within milliseconds
2. **Consistency Guarantees**: All nodes see the same configuration
3. **Automatic Conflict Resolution**: Timestamps and version vectors prevent conflicts
4. **Local Caching**: Each node maintains a complete copy in RAM

### Resource Scheduling and Load Balancing

The cluster scheduler considers multiple factors:

- Current resource utilization (CPU, RAM, storage)
- Network topology and latency
- Storage availability and performance
- Administrator-defined policies
- Historical performance data

## 1.4 New Features in Proxmox VE 9

The release of Proxmox VE 9 on August 5, 2025, marks a watershed moment in the platform's evolution. This version addresses long-standing enterprise requirements while introducing innovative features that position Proxmox VE at the forefront of virtualization technology.

## Foundation Updates: Building on Debian 13 "Trixie"

The decision to base Proxmox VE 9 on Debian 13 "Trixie" provides immediate benefits:

### Kernel 6.14.8 Enhancements

The new kernel brings substantial improvements:

1. **Hardware Support**:
    - PCIe 5.0 with speeds up to 32 GT/s per lane
    - DDR5 memory controller optimizations
    - Intel Sapphire Rapids and AMD Genoa CPU features
    - Enhanced GPU passthrough for NVIDIA and AMD
2. **Performance Optimizations**:
    - Core scheduling for better CPU security
    - Improved NUMA balancing algorithms
    - Enhanced memory tiering support
    - Optimized interrupt handling for high-speed networks
3. **Security Features**:
    - Kernel Control Flow Integrity (CFI)
    - Enhanced randomization of kernel structures
    - Improved mitigation for speculative execution vulnerabilities

### Component Version Matrix

| Component | Proxmox VE 8.x | Proxmox VE 9.0 | Key Improvements |
|---|---|---|---|
| **Kernel** | 6.2.x | 6.14.8 | PCIe 5.0, DDR5, Security |
| **QEMU** | 8.0.x | 10.0.2 | Performance, Migration |
| **LXC** | 5.0.x | 6.0.4 | Cgroup v2, Security |
| **ZFS** | 2.1.x | 2.3.3 | RAID-Z expansion |
| **Ceph** | Reef 18.x | Squid 19.2.3 | Performance, Features |

## Revolutionary Storage Features

The storage enhancements in Proxmox VE 9 address critical enterprise requirements that have long been pain points for organizations with existing storage infrastructure.

### Snapshots for Thick-Provisioned LVM: A Game Changer

The implementation of snapshots for thick-provisioned LVM storage represents a fundamental shift in how Proxmox VE handles traditional enterprise storage. This feature is particularly significant for organizations with substantial investments in:

- Fibre Channel SANs from vendors like EMC, NetApp, or HPE
- iSCSI arrays that don't support thin provisioning
- Existing LVM-based storage infrastructure

**Technical Implementation:**

The snapshot mechanism uses a volume chain approach:

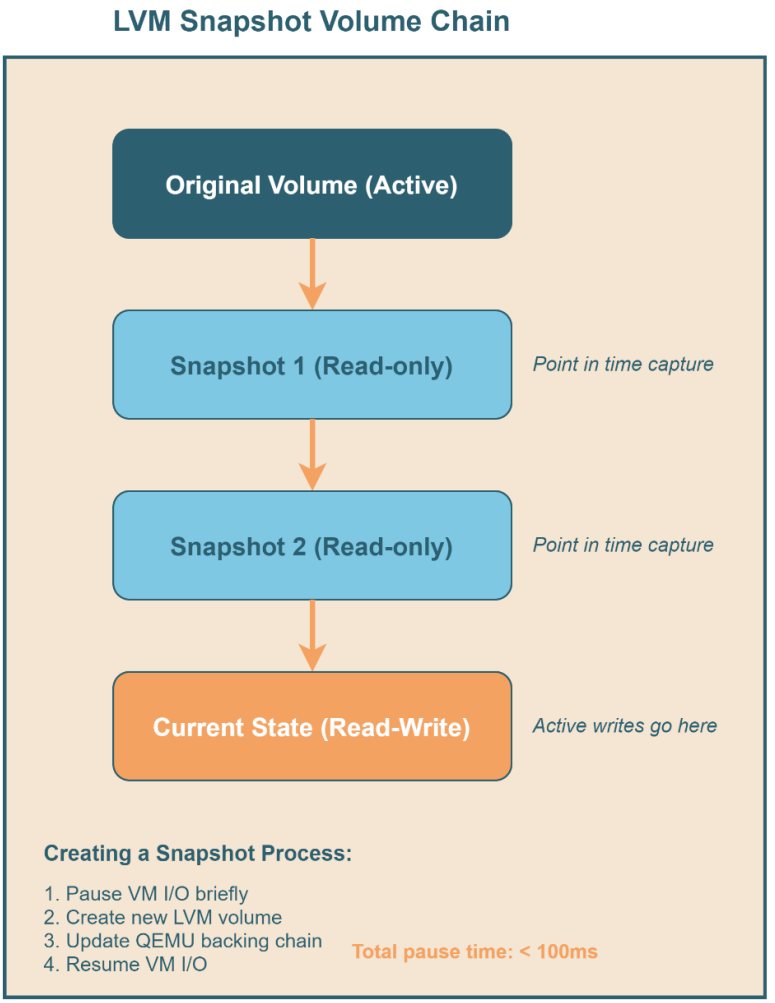



*Figure 8 LVM Snapshot Volume Chain*

Each snapshot captures the state at a point in time, with new writes going to a new volume:

1. **Creating a Snapshot:**

*# Internal process flow*

1. Pause VM I/O briefly
2. Create new LVM volume
3. Update QEMU backing chain
4. Resume VM I/O

*# Total pause time: <100ms*

2. **Storage Efficiency:**
    - Only changed blocks consume additional space
    - Metadata tracking via qcow2 format
    - Automatic cleanup of unused blocks
3. **Performance Characteristics:**
    - Minimal impact on running VMs
    - Sequential read performance maintained
    - Small penalty for random writes due to COW

**ZFS RAID-Z Expansion: Long-Awaited Flexibility**

The ability to expand RAID-Z vdevs addresses a limitation that has existed since ZFS's inception. This feature, arriving with ZFS 2.3.3, fundamentally changes capacity planning for ZFS deployments:

**Expansion Process:**

```
# Before expansion: 4-disk RAID-Z1
tank     ONLINE       0     0     0
  raidz1-0  ONLINE    0     0     0
    sda     ONLINE    0     0     0
    sdb     ONLINE    0     0     0
    sdc     ONLINE    0     0     0
    sdd     ONLINE    0     0     0

# Add new disk to existing RAID-Z
zpool attach tank raidz1-0 sde

# After expansion: 5-disk RAID-Z1
tank     ONLINE       0     0     0
  raidz1-0  ONLINE    0     0     0
    sda     ONLINE    0     0     0
    sdb     ONLINE    0     0     0
    sdc     ONLINE    0     0     0
    sdd     ONLINE    0     0     0
    sde     ONLINE    0     0     0
```

**Considerations:**

- Rebalancing happens automatically in the background

- Existing data is redistributed across all disks
- Pool remains online throughout the process
- New capacity becomes available immediately

## Advanced Networking with SDN Fabrics

The SDN Fabrics feature transforms Proxmox VE into a platform capable of building complex network topologies that previously required dedicated network hardware or specialized software.

### Understanding SDN Fabrics

SDN Fabrics provide automated configuration of underlay networks for complex topologies:

Spine-Leaf Architecture with SDN Fabrics

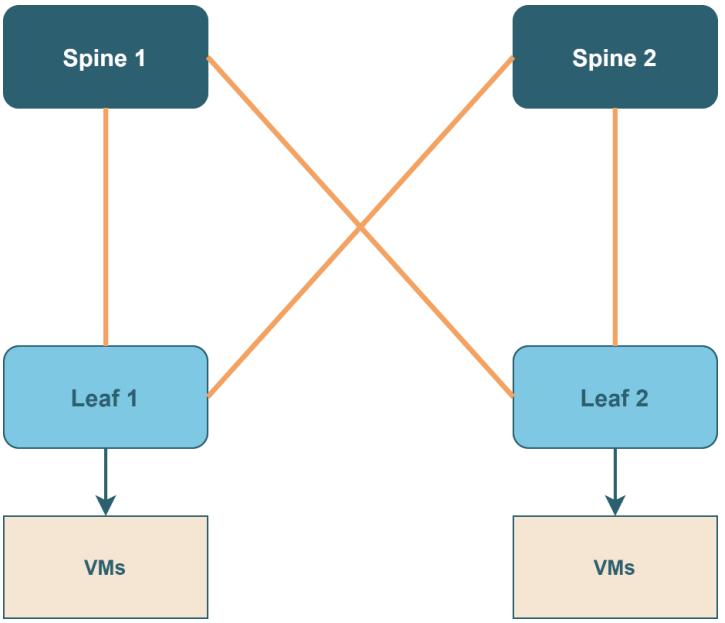



*Figure 9 Spine-Leaf Architecture with SDN Fabrics*

**Key Capabilities:**

1. **Dynamic Routing Integration:**
    - OSPF for traditional IP routing
    - BGP for large-scale deployments
    - OpenFabric for automatic topology discovery
2. **Automated Configuration:**

- Template-based deployment
- Automatic IP assignment
- Route propagation without manual intervention

3. **Use Cases**:
   - **Ceph Networks**: Full-mesh connectivity between storage nodes
   - **EVPN Underlays**: Foundation for overlay networks
   - **Multi-Site Connectivity**: Stretched clusters across locations

**Configuration Example**:

```
fabric: ceph-mesh
    type: open-fabric
    asn: 65000
    nodes:
        - name: node1
          router-id: 10.0.0.1
          interfaces:
            - eth2: 10.0.0.1/24
        - name: node2
          router-id: 10.0.0.2
          interfaces:
            - eth2: 10.0.0.2/24
    auto-mesh: true
```

## High Availability Evolution: Affinity Rules

The new HA affinity rules system represents a complete reimagining of how Proxmox VE handles resource placement in clusters. This feature addresses complex requirements for both performance optimization and compliance.

### Node Affinity: Controlling VM Placement

Node affinity rules allow precise control over where VMs run:

Rule Types:

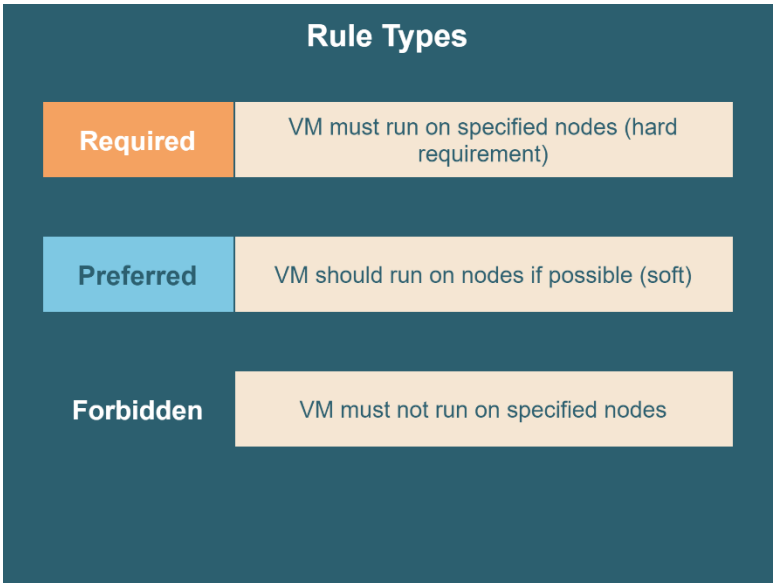



*Figure 10 HA Affinity Rules Types*

### Resource Affinity: Managing VM Relationships

Resource affinity defines relationships between VMs:

1. **Affinity Groups** (Keep Together):
    - Database and application tiers on same node
    - Reduces network latency
    - Improves cache efficiency
2. **Anti-Affinity Groups** (Keep Apart):
    - Redundant services on different nodes
    - Compliance with failure domain requirements
    - Load distribution across infrastructure

### Migration from HA Groups

Existing HA groups are automatically converted to affinity rules:

```
# Old HA Group Configuration
group: db-servers
    nodes node1,node2
    restricted 1

# Converted to Affinity Rule
affinity-rule: db-servers
    type: node-required
    nodes: node1,node2
    vms: 101,102,103
```

## Modernized Mobile Interface

The complete rewrite of the mobile interface using Rust and the Yew framework represents Proxmox's commitment to modern development practices and superior user experience.

**Architecture Benefits:**

1. **Performance:**
    - Compiled WebAssembly for near-native speed
    - Minimal JavaScript overhead
    - Efficient DOM updates
2. **Reliability:**
    - Type-safe code prevents runtime errors
    - Memory safety guarantees from Rust
    - Comprehensive error handling
3. **User Experience:**
    - Responsive design adapts to screen size
    - Touch-optimized controls
    - Offline capability for basic operations

**Key Features:**

Mobile Interface Capabilities


*Figure 11 Mobile Interface Capabilities*

## Enhanced Monitoring and Metrics

The new metrics system provides unprecedented visibility into system behavior, essential for troubleshooting and capacity planning.

### Pressure Stall Information (PSI)

PSI metrics reveal when systems are under resource pressure:

CPU Pressure: some=12.5% full=2.1%

├─ "some": At least one task delayed

└─ "full": All non-idle tasks delayed

Memory Pressure: some=8.3% full=0.5%

├─ Indicates memory allocation delays

└─ Critical for identifying OOM risks

I/O Pressure: some=15.2% full=3.8%

├─ Storage subsystem bottlenecks

└─ Helps identify slow storage

**ZFS ARC Visibility**

The ZFS ARC (Adaptive Replacement Cache) metrics integration provides:

Node Memory Usage

├─ Total: 64 GB

├─ Used: 45 GB

│ ├─ VMs/CTs: 32 GB

│ ├─ ZFS ARC: 10 GB

│ └─ System: 3 GB

└─ Free: 19 GB

This visibility helps administrators:

- Right-size ZFS ARC limits
- Understand memory pressure sources
- Optimize overall memory allocation

## 1.5 Book Overview and Lab Environment

This book is structured as a journey from foundational concepts to advanced deployment scenarios. Each chapter builds upon previous knowledge while introducing new concepts and techniques. The hands-on approach ensures that theoretical knowledge is immediately reinforced through practical application.

### Learning Path and Book Structure

The book follows a carefully designed learning path that mirrors real-world deployment scenarios:

**Progressive Complexity Model**

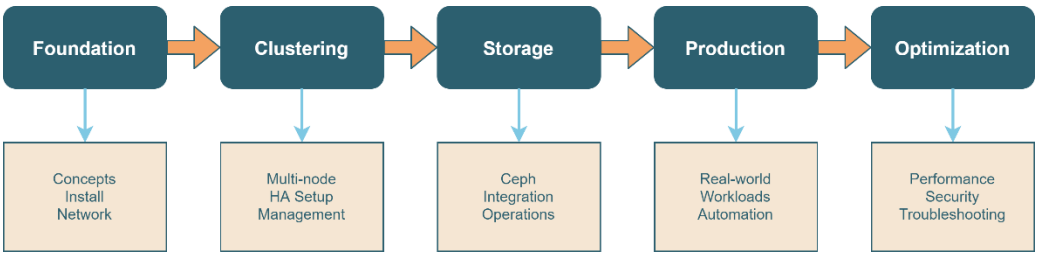



*Figure 12 Progressive Complexity Model*

**Part I: Foundation (Chapters 1-3)**

This section establishes the groundwork:

- **Chapter 1**: Comprehensive introduction to Proxmox VE architecture and capabilities
- **Chapter 2**: Detailed lab environment setup with KVM nested virtualization
- **Chapter 3**: First node installation and initial configuration

Learning outcomes:

- Understand Proxmox VE's position in the virtualization landscape
- Configure a complete lab environment
- Perform basic installation and setup

**Part II: Building the Cluster (Chapters 4-6)**

Clustering transforms standalone nodes into a unified platform:

- **Chapter 4**: Creating and managing multi-node clusters
- **Chapter 5**: Implementing high availability and advanced features
- **Chapter 6**: Storage architecture and integration options

Learning outcomes:

- Design and implement resilient clusters
- Configure automatic failover
- Understand storage abstraction layers

**Part III: Implementing Ceph Storage (Chapters 7-10)**

The Ceph section represents the book's technical apex:

- **Chapter 7**: Deep dive into Ceph architecture and concepts
- **Chapter 8**: Step-by-step Ceph deployment on Proxmox VE
- **Chapter 9**: Advanced configuration and performance tuning

- **Chapter 10**: Operational procedures and maintenance

Learning outcomes:

- Master distributed storage concepts
- Deploy production-ready Ceph clusters
- Optimize performance for various workloads

**Part IV: Production Deployment (Chapters 11-13)**

Transitioning from lab to production:

- **Chapter 11**: Virtual machine lifecycle management
- **Chapter 12**: Container deployment and orchestration
- **Chapter 13**: Automation and integration strategies

Learning outcomes:

- Manage complex VM environments
- Implement container strategies
- Automate routine operations

**Part V: Best Practices and Troubleshooting (Chapters 14-15)**

Ensuring long-term success:

- **Chapter 14**: Security hardening and compliance
- **Chapter 15**: Troubleshooting methodologies and tools

Learning outcomes:

- Implement security best practices
- Diagnose and resolve complex issues
- Optimize for specific workloads

## Lab Environment Architecture

The lab environment is designed to provide a realistic yet accessible platform for learning Proxmox VE. Using nested virtualization on a single physical host, we create a complete three-node cluster with full Ceph integration.

### Physical Host Requirements

The physical host, designated as "lab1", serves as the foundation:

Minimum Specifications:

```
┌──────────────────────────────────┐
│ CPU: 8+ cores with VT-x/AMD-V    │
│ RAM: 64GB (96GB recommended)     │
│ Storage: 500GB SSD               │
│ Network: Gigabit Ethernet        │
│ OS: Ubuntu 22.04 LTS             │
└──────────────────────────────────┘
```

Recommended Specifications:

```
┌──────────────────────────────────┐
│ CPU: 16+ cores, dual socket      │
│ RAM: 128GB ECC                   │
│ Storage: 1TB NVMe + 2TB SSD      │
│ Network: 10Gb Ethernet           │
│ OS: Ubuntu 22.04 LTS             │
└──────────────────────────────────┘
```

### Nested Virtualization Configuration

Enabling nested virtualization requires specific configuration:

```
# Intel systems
echo "options kvm_intel nested=1" > /etc/modprobe.d/kvm.conf
```

```
# AMD systems
echo "options kvm_amd nested=1" > /etc/modprobe.d/kvm.conf
```

```
# Verify nested virtualization
cat /sys/module/kvm_intel/parameters/nested
Y
```

### KVM and libvirt Setup

The complete virtualization stack installation:

```
# Install KVM and management tools
apt update
apt install -y qemu-kvm libvirt-daemon-system \
    libvirt-clients bridge-utils virt-manager

# Install Open vSwitch for advanced networking
apt install -y openvswitch-switch
```

```
# Add user to libvirt group
usermod -aG libvirt $USER

# Verify installation
virsh version
systemctl status libvirtd
ovs-vsctl show
```

## Network Design and Implementation

The network architecture simulates a production environment with proper segmentation:

### Network Topology Overview

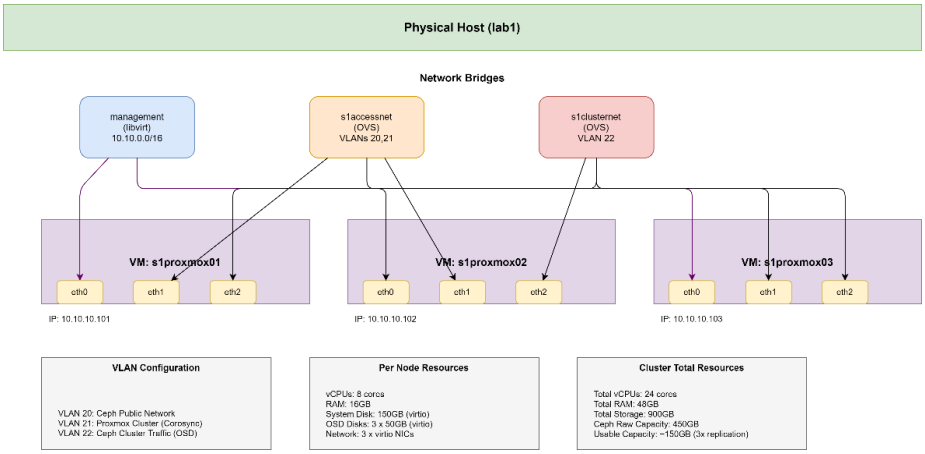



*Figure 13 Lab Network Topology Overview*

### Management Network (10.10.0.0/16)

Purpose and configuration:

- External connectivity via NAT
- Management access to Proxmox VE nodes
- Software updates and internet access

```
<network>
  <name>management</name>
  <bridge name="management"/>
  <forward mode="nat" dev='eno1'/>
  <ip address="10.10.0.1" netmask="255.255.0.0">
    <dhcp>
      <range start="10.10.1.1" end="10.10.254.254"/>
    </dhcp>
  </ip>
</network>
```

### Access Network (s1accessnet)

Multi-purpose network with VLAN segregation:

- VLAN 20: Ceph public network (client access)
- VLAN 21: Proxmox cluster communication (Corosync)

```
<network>
  <name>s1accessnet</name>
  <forward mode='bridge'/>
  <bridge name='s1accessnet'/>
  <virtualport type='openvswitch'/>
</network>
```

Open vSwitch configuration:

```
# Create OVS bridge
ovs-vsctl add-br s1accessnet

# Configure VLANs
ovs-vsctl set port s1accessnet tag=20,21

# Verify configuration
ovs-vsctl show
```

### Cluster Network (s1clusternet)

Dedicated Ceph cluster traffic:

- VLAN 22: Ceph OSD replication
- Isolated from other traffic
- High bandwidth, low latency

```
<network>
  <name>s1clusternet</name>
  <forward mode='bridge'/>
  <bridge name='s1clusternet'/>
  <virtualport type='openvswitch'/>
</network>
```

## Virtual Node Specifications

Each Proxmox VE node is carefully configured to support both learning and realistic workloads:

### Resource Allocation

Per Node Configuration:

```
┌──────────────────────────────────┐
│ vCPUs: 8 cores                   │
│ RAM: 16GB                        │
│ System Disk: 150GB (virtio)      │
│ OSD Disks: 3 x 50GB (virtio)     │
│ Network: 3 x virtio NICs         │
│ Display: VNC (unique ports)      │
└──────────────────────────────────┘
```

Aggregate Cluster Resources:

```
┌──────────────────────────────┐
│ Total vCPUs: 24 cores        │
│ Total RAM: 48GB              │
│ Total Storage: 900GB         │
│ Ceph Raw Capacity: 450GB     │
│ Usable Capacity: ~150GB (3x) │
└──────────────────────────────┘
```

### VM Creation Commands

The virt-install commands create properly configured VMs:

```
# Node 1 creation
virt-install --name s1proxmox01 \
  --ram 16384 \
  --disk path=/var/lib/libvirt/images/s1proxmox01.img,size=150 \
  --vcpus 8 \
  --network network:management \
  --network bridge=s1accessnet,\
    mac=52:54:00:31:7d:87,\
    virtualport_type=openvswitch,\
    model=virtio,driver.name=vhost \
  --network bridge=s1clusternet,\
    mac=52:54:00:46:59:08,\
    virtualport_type=openvswitch,\
    model=virtio,driver.name=vhost \
  --console pty,target_type=serial \
  --cdrom /path/to/proxmox-ve-9.0.iso \
  --graphics vnc,listen=0.0.0.0,port=6401,keymap=fr
```

```
# Add OSD disks post-installation
virsh attach-disk s1proxmox01 \
  --source /var/lib/libvirt/images/s1proxmox01-osd1.img \
  --target vdb --persistent --size 50
```

```
virsh attach-disk s1proxmox01 \
  --source /var/lib/libvirt/images/s1proxmox01-osd2.img \
  --target vdc --persistent --size 50
```

```
virsh attach-disk s1proxmox01 \
  --source /var/lib/libvirt/images/s1proxmox01-osd3.img \
  --target vdd --persistent --size 50
```

## Lab Exercise Structure

Each chapter includes structured exercises designed to reinforce learning:

### Exercise Types

1. **Guided Labs**: Step-by-step instructions
    - Clear objectives
    - Detailed procedures
    - Expected outcomes
    - Verification steps
2. **Challenge Labs**: Problem-solving scenarios
    - Business requirements
    - Technical constraints
    - Multiple valid solutions
    - Real-world applicability
3. **Troubleshooting Labs**: Diagnostic exercises
    - Simulated failures
    - Systematic diagnosis
    - Resolution procedures
    - Prevention strategies

**Example Exercise Format**

Lab 3.1: Initial Proxmox VE Installation

---

Objective: Install Proxmox VE on first node

Prerequisites:

- VM s1proxmox01 created and powered on
- Access to VNC console (10.10.0.1:6401)
- Proxmox VE 9.0 ISO mounted

Estimated Time: 45 minutes

Steps:

1. Connect to VNC console

2. Select "Install Proxmox VE"

3. Configure installation:

- Disk: /dev/vda (150GB)

- Country/Timezone: Your location

- Password: Strong password
- Email: admin@lab.local
- Network:
  - IP: 10.10.10.101/24
  - Gateway: 10.10.0.1
  - DNS: 10.10.0.1

Verification:

- Web UI accessible at https://10.10.10.101:8006
- Login successful with root credentials
- All services running: systemctl status

Common Issues:

- Network misconfiguration: Check IP/Gateway
- DNS resolution: Verify /etc/resolv.conf
- Service failures: Review systemctl logs

## Summary

This chapter has provided a comprehensive introduction to Proxmox VE 9, covering its architecture, capabilities, and the significant enhancements in the latest release. We've explored why Proxmox VE represents a compelling alternative to traditional virtualization platforms and outlined the learning journey ahead.

The lab environment we'll build provides a realistic platform for mastering Proxmox VE without requiring extensive hardware resources. Through hands-on exercises and real-world scenarios, you'll develop the skills needed to deploy and manage production Proxmox VE infrastructure.

### Key Takeaways

- Proxmox VE integrates KVM and LXC virtualization with enterprise management features
- Version 9.0 introduces critical features for enterprise adoption, including LVM snapshots and SDN fabrics
- The platform provides cost-effective virtualization without sacrificing capabilities
- Open standards ensure no vendor lock-in and future flexibility
- Our lab environment simulates production scenarios using nested virtualization

## What's Next

In Chapter 2, we'll dive into the practical aspects of building your lab environment. You'll learn to:

- Configure Ubuntu 22.04 as a virtualization host
- Set up KVM with nested virtualization support
- Create Open vSwitch networks with VLAN tagging
- Deploy the three Proxmox VE virtual nodes
- Prepare the infrastructure for cluster creation

Prepare to transform your physical server into a complete virtualization lab that will serve as your learning platform throughout this book. The journey from concept to implementation begins now.

# Chapter 2: Building Your Virtualization Lab

In this chapter, we transition from theory to practice, building a complete virtualization lab that will serve as our learning environment throughout this book. Using KVM (Kernel-based Virtual Machine) on a physical host, we'll create a sophisticated nested virtualization setup that accurately simulates a production Proxmox VE deployment. This hands-on approach ensures you gain practical experience with every aspect of Proxmox VE, from initial installation through advanced clustering and storage configuration.

## 2.1 Lab Architecture Overview

Before diving into the technical implementation, it's essential to understand the architecture we're building. Our lab design mirrors real-world deployments while remaining feasible on a single physical server. This approach provides authentic learning experiences without requiring expensive hardware investments.

### Understanding the Lab Design

Our lab architecture consists of several interconnected components that work together to create a realistic virtualization environment:

Physical Host (lab1)
├── Host Operating System: Linux (with KVM support)
├── Hypervisor: KVM with QEMU
├── Network Virtualization: Open vSwitch
├── Management Tools: libvirt, virsh
└── Virtual Infrastructure
  ├── Management Network (10.10.0.0/16)
  │  └── Gateway: 10.10.0.1
  ├── Node 1: s1proxmox01 (10.10.44.1)
  ├── Node 2: s1proxmox02 (10.10.44.2)
  ├── Node 3: s1proxmox03 (10.10.44.3)
  ├── OVS Networks
  │  ├── s1accessnet (VLANs 20, 21)
  │  └── s1clusternet (VLAN 22)

└── Support VM: Ubuntu Desktop 24.04

This nested virtualization approach—running Proxmox VE as virtual machines on top of KVM—provides several advantages:

**Flexibility**: Easy to create, destroy, and reconfigure nodes without affecting physical hardware. This is particularly valuable when learning, as mistakes can be quickly corrected without lasting consequences.

**Cost-Effectiveness**: A single physical server can simulate an entire cluster, dramatically reducing hardware requirements while maintaining realistic behavior.

**Snapshot Capability**: The ability to snapshot entire nodes before major changes provides a safety net for experimentation.

**Network Isolation**: Complete control over network topology without affecting production networks or requiring physical network changes.

## Resource Planning

Proper resource allocation is crucial for a functional lab environment. Here's our resource distribution for the three-node cluster:

**Per Proxmox VE Node**:

- RAM: 16 GB
- vCPUs: 8 cores
- System Disk: 150 GB
- Ceph OSD Disks: 3 × 50 GB

**Total Lab Requirements**:

- RAM: 48 GB (nodes) + 8 GB (desktop) + 8 GB (host overhead) = 64 GB minimum
- CPU: 24 vCPUs + overhead (recommend 32+ physical cores)
- Storage: 450 GB (nodes) + 150 GB (Ceph) + 40 GB (desktop) = 640 GB minimum

**Network Architecture and IP Addressing Plan**:

| Network | VLAN | Subnet | Gateway | Description |
|---|---|---|---|---|
| **Management** | None | 10.10.0.0/16 | 10.10.0.1 | External access, Internet connectivity |
| **Ceph Public** | 20 | 10.20.20.0/24 | None | Client access to Ceph storage |
| **Proxmox Cluster** | 21 | 10.21.21.0/24 | None | Corosync cluster communication |
| **Ceph Cluster** | 22 | 10.22.22.0/24 | None | OSD replication traffic |

**Node IP Assignments**:

| Node | Management | VLAN 20 (Ceph Public) | VLAN 21 (Cluster) | VLAN 22 (Ceph Cluster) |
|---|---|---|---|---|
| **s1proxmox01** | 10.10.44.1/16 | 10.20.20.1/24 | 10.21.21.1/24 | 10.22.22.1/24 |
| **s1proxmox02** | 10.10.44.2/16 | 10.20.20.2/24 | 10.21.21.2/24 | 10.22.22.2/24 |
| **s1proxmox03** | 10.10.44.3/16 | 10.20.20.3/24 | 10.21.21.3/24 | 10.22.22.3/24 |
| **Desktop VM** | DHCP | N/A | N/A | N/A |

**Network Interface Mapping:**

| Node Interface | Connected To | Purpose |
|---|---|---|
| **ens3** | management | Management network (vmbr0) |
| **ens4** | s1accessnet | VLANs 20 & 21 (vmbr1) |
| **ens5** | s1clusternet | VLAN 22 (vmbr2) |

## 2.2 Preparing the KVM Host

The foundation of our lab is a properly configured KVM host. This section covers the essential preparations needed before deploying Proxmox VE nodes.

### Installing Essential Packages

Start by installing the required virtualization packages:

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install KVM and related tools
sudo apt install -y qemu-kvm libvirt-daemon-system libvirt-clients \
  bridge-utils virt-manager virtinst cpu-checker

# Verify KVM installation
kvm-ok
# Expected output: INFO: /dev/kvm exists

# Add your user to necessary groups
sudo usermod -aG libvirt,kvm $USER

# Log out and back in for group changes to take effect
```

### Enabling Nested Virtualization

Nested virtualization allows our Proxmox VE VMs to run their own virtual machines. This feature must be explicitly enabled:

**For Intel CPUs:**

```
# Check current status
cat /sys/module/kvm_intel/parameters/nested
```

```
# Enable nested virtualization
echo "options kvm_intel nested=1" | sudo tee /etc/modprobe.d/kvm.conf

# Reload the module
sudo modprobe -r kvm_intel
sudo modprobe kvm_intel

# Verify the change
cat /sys/module/kvm_intel/parameters/nested
# Should output: Y
```

**For AMD CPUs:**

```
# Enable for AMD
echo "options kvm_amd nested=1" | sudo tee /etc/modprobe.d/kvm.conf

# Reload the module
sudo modprobe -r kvm_amd
sudo modprobe kvm_amd
```

**Persistent Configuration:**

To ensure nested virtualization survives reboots:

```
# Update initramfs
sudo update-initramfs -u

# Verify after reboot
sudo reboot
# After reboot:
cat /sys/module/kvm_intel/parameters/nested
```

## Installing Open vSwitch

Open vSwitch (OVS) provides advanced networking features, including VLAN support, which is crucial for our lab's network design:

```
# Install Open vSwitch
sudo apt install -y openvswitch-switch

# Verify installation
sudo ovs-vsctl show

# Check service status
systemctl status openvswitch-switch
```

**Basic OVS Configuration:**

```
# View OVS version
ovs-vsctl --version

# List all bridges (initially empty)
ovs-vsctl list-br

# Show detailed configuration
ovs-vsctl show
```

# 2.3 Designing the Network Infrastructure

Network design is critical for a functional Proxmox VE cluster. Our lab implements a production-like network topology using software-defined networking principles. To be realistic, we'll use VLAN-capable vSwitches to segment different types of traffic, mirroring enterprise deployments.

## Understanding the Network Architecture

Our lab uses three distinct networks, each serving specific purposes:

Network Overview:

1. management (10.10.0.0/16) - No VLAN
   - NAT network for external/internet access
   - Management access to nodes
   - Gateway: 10.10.0.1

2. s1accessnet - VLAN-capable OVS bridge
   - VLAN 20: Ceph Public Network (10.20.20.0/24)
   - VLAN 21: Proxmox Cluster Network (10.21.21.0/24)

3. s1clusternet - VLAN-capable OVS bridge
   - VLAN 22: Ceph Cluster Network (10.22.22.0/24)

Why This Design?

This architecture provides several benefits:

1. **Traffic Segregation**: Different traffic types don't compete for bandwidth
2. **Security**: Cluster and storage traffic are isolated from management
3. **Realistic Setup**: Mirrors production environments with VLAN segmentation
4. **Performance**: Dedicated networks prevent congestion
5. **Learning Value**: Hands-on experience with enterprise network design

## VLAN Assignments and Purpose

Each VLAN serves a specific purpose in our cluster:

VLAN 20 - Ceph Public Network (10.20.20.0/24):

- Client access to Ceph storage
- Communication between Proxmox VE and Ceph monitors
- Storage API traffic

**VLAN 21 - Proxmox Cluster Network (10.21.21.0/24):**

- Corosync cluster communication
- High-priority, low-latency traffic
- Cluster state synchronization
- Migration traffic

**VLAN 22 - Ceph Cluster Network (10.22.22.0/24):**

- OSD-to-OSD replication traffic
- Recovery and rebalancing operations
- Isolated from client traffic for performance

## Creating the Networks

We'll create each network using libvirt network definitions and Open vSwitch:

**Step 1: Create the Management Network**

First, create the network definition file:

```
cat > management.xml << 'EOF'
<network>
  <name>management</name>
  <bridge name="management"/>
  <forward mode="nat" dev='eno1'/>
  <ip address="10.10.0.1" netmask="255.255.0.0">
  </ip>
</network>
EOF
```

Define and start the network:

```
# Define the network in libvirt
virsh net-define management.xml

# Start the network
virsh net-start management

# Enable autostart
virsh net-autostart management

# Verify
virsh net-info management
```

**Step 2: Create the Access Network (s1accessnet)**

Create the OVS bridge:

```
# Create Open vSwitch bridge
sudo ovs-vsctl add-br s1accessnet
```

Create the libvirt network definition:

```
cat > s1accessnet-network.xml << 'EOF'
<network>
  <name>s1accessnet</name>
  <forward mode='bridge'/>
  <bridge name='s1accessnet'/>
  <virtualport type='openvswitch'/>
</network>
EOF
```

Define and start the network:

```
# Define the network
virsh net-define s1accessnet-network.xml

# Start the network
virsh net-start s1accessnet
```

*# Note: We'll enable autostart after all networks are tested*

### Step 3: Create the Cluster Network (s1clusternet)

Create the OVS bridge:

```
# Create Open vSwitch bridge
sudo ovs-vsctl add-br s1clusternet
```

Create the libvirt network definition:

```
cat > s1clusternet-network.xml << 'EOF'
<network>
  <name>s1clusternet</name>
  <forward mode='bridge'/>
  <bridge name='s1clusternet'/>
  <virtualport type='openvswitch'/>
</network>
EOF
```

Define and start the network:

```
# Define the network
virsh net-define s1clusternet-network.xml

# Start the network
virsh net-start s1clusternet
```

### Step 4: Verify All Networks

Check that all networks are properly created:

```
# List all networks
virsh net-list

# Expected output:
```

```
 Name              State    Autostart   Persistent
----------------------------------------------------
 management        active   yes         yes
 s1accessnet       active   no          yes
 s1clusternet      active   no          yes

# Verify OVS bridges
sudo ovs-vsctl show

# Should show both s1accessnet and s1clusternet bridges
```

## Making Networks Persistent

To ensure networks survive host reboots:

### Enable Autostart for Virtual Networks:

```
# Set networks to autostart
virsh net-autostart s1accessnet
virsh net-autostart s1clusternet

# Verify autostart is enabled
virsh net-list --all
```

### Create Systemd Service for OVS Bridges:

```
# Create service to ensure OVS bridges are up
cat << 'EOF' | sudo tee /etc/systemd/system/ovs-lab-networks.service
[Unit]
Description=OVS Lab Networks for Proxmox
After=network.target openvswitch-switch.service
Requires=openvswitch-switch.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/bin/ovs-vsctl --may-exist add-br s1accessnet
ExecStart=/usr/bin/ovs-vsctl --may-exist add-br s1clusternet
ExecStart=/usr/bin/ip link set s1accessnet up
ExecStart=/usr/bin/ip link set s1clusternet up
ExecStop=/usr/bin/ip link set s1accessnet down
ExecStop=/usr/bin/ip link set s1clusternet down

[Install]
WantedBy=multi-user.target
EOF

# Enable and start the service
sudo systemctl enable ovs-lab-networks.service
sudo systemctl start ovs-lab-networks.service

# Check service status
sudo systemctl status ovs-lab-networks.service
```

## Network Architecture Diagram

The complete network topology for our three-node cluster:

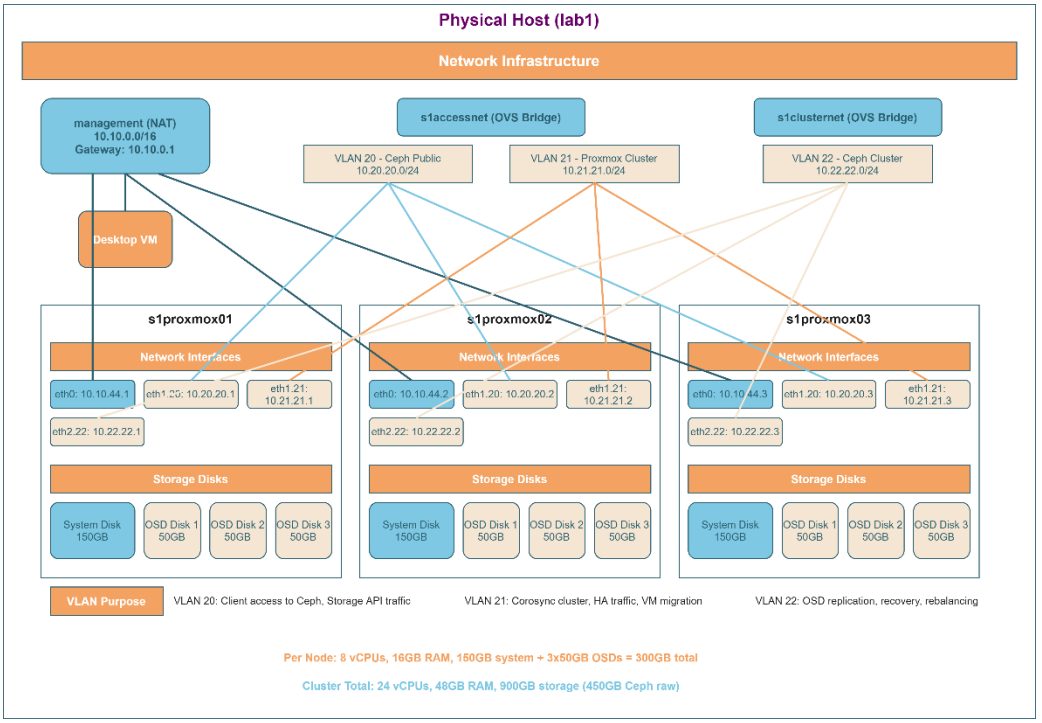



*Figure 14 Network Architecture Diagram*

This network design ensures proper traffic isolation while maintaining the flexibility needed for a learning environment.

# 2.4 Installing Proxmox VE Nodes

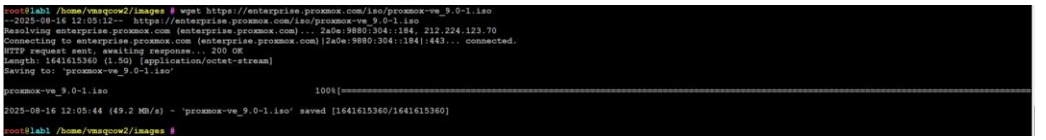
With the infrastructure ready, we can now deploy our three Proxmox VE nodes. Each installation follows the same pattern but with unique network configurations.

## Downloading Proxmox VE 9

First, obtain the Proxmox VE ISO image:

```
# Create directory for ISOs
mkdir -p ~/iso

# Download Proxmox VE 9.0 ISO (adjust version as needed)
cd ~/iso
wget https://www.proxmox.com/images/download/pve/iso/proxmox-ve_9.0-1.iso
```

```
# Verify download (optional but recommended)
# Check SHA256 sum from Proxmox website
sha256sum proxmox-ve_9.0-1.iso
```




## Creating the First Node

Deploy the first Proxmox VE node using virt-install:

```
virt-install --name s1proxmox01 \
  --ram 16384 \
  --disk path=/var/lib/libvirt/images/s1proxmox01.img,size=150 \
  --vcpus 8 \
  --network network:management \
  --network bridge=s1accessnet,mac=52:54:00:31:7d:87,virtualport_type=openvswitch,model=virtio,driver.name=vhost \
  --network bridge=s1clusternet,mac=52:54:00:46:59:08,virtualport_type=openvswitch,model=virtio,driver.name=vhost \
  --console pty,target_type=serial \
  --cdrom /home/vmsqcow2/images/proxmox-ve_9.0-1.iso \
  --graphics vnc,listen=0.0.0.0,port=60100,keymap=fr
```

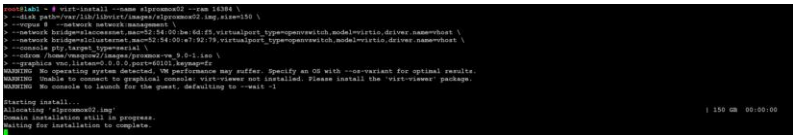
**Note**: The VNC port (60100) will be used to access the installer graphical interface. You can generate unique MAC addresses using:

```
date +%s | md5sum | head -c 6 | sed -e 's/\([0-9A-Fa-f]\{2\}\)/\1:/g' -e 's/\(.*\):$/\1/' | sed -e
's/^/52:54:00:/'
```

## Installation Walkthrough

After launching the virt-install command, you'll see output similar to:

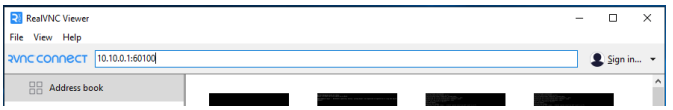
```
WARNING  No operating system detected, VM performance may suffer. Specify an OS with --os-variant for optimal results.
WARNING  Unable to connect to graphical console: virt-viewer not installed. Please install the 'virt-viewer' package.
WARNING  No console to launch for the guest, defaulting to --wait -1

Starting install...
Allocating 's1proxmox01.img'                                  | 150 GB  00:00:00
Domain installation still in progress.
Waiting for installation to complete.
```



Now, connect to the VNC console from your desktop machine using a VNC viewer (such as TigerVNC, RealVNC, or TightVNC):

### Step 1: Connect via VNC

- Open your VNC viewer application
- Connect to: <host-ip>:60100 (e.g., 10.10.0.1:60100)



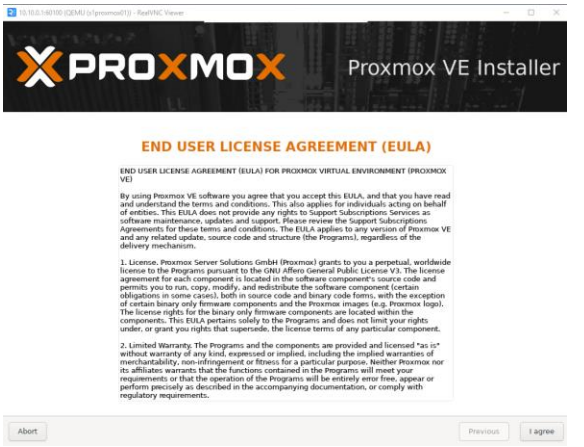

- You should see the Proxmox VE installer boot screen

### Step 2: Start Installation

- Press Enter on "Install Proxmox VE (Graphical)"
- The installer will load the graphical interface


### Step 3: License Agreement

- Read through the End User License Agreement
- Click "I agree" to proceed




### Step 4: Target Hard Disk Selection

- The installer will detect available disks
- Select /dev/sda (the 150GB disk we created)
- Leave the default options unless you need specific configurations
- Click "Next"