NexaSphere

Technical Whitepaper

Version: V2.1.1

Issue: V2.1.1

Nexavm Technologies AG

NEXAVM

Copyright Statement

Copyright © 2026 Nexavm Technologies AG. All rights reserved.

No part of this document may be excerpted, reproduced, or distributed in any form without prior written permission from Nexavm Technologies AG.

Trademark Notice

NexaSphere and other trademarks of Nexavm Technologies AG are trademarks of Nexavm Technologies AG.

All other trademarks or registered trademarks mentioned in this document are the property of their respective owners.

Notice

The products, services, and features you purchase are governed by the applicable commercial contracts and terms of Nexavm Technologies AG. Some products, services, or features described in this document might not be included in your purchase or available for your use. Unless otherwise agreed in the applicable contract, Nexavm Technologies AG makes no express or implied representations or warranties regarding the contents of this document.

This document might be updated from time to time due to product version upgrades or other reasons. Unless otherwise agreed, this document is provided for guidance only. The statements , information, and recommendations in this document do not constitute any express or implied warranty.

Contents

Copyright Statement..... I
Technical Whitepaper.....1
1.1 Overview.....1
1.2 Technical Advantages.....3
1.3 Product Architecture.....4
1.3.1 Software Architecture.....4
1.3.2 Functional Architecture.....6
1.3.3 Resource Model.....7
1.4 Core Design.....10
1.4.1 Resource Virtualization.....10
1.4.1.1 Compute Virtualization.....10
1.4.1.1.1 Overview.....10
1.4.1.1.2 Key Features.....11
1.4.1.1.2.1 CPU Virtualization.....11
1.4.1.1.2.2 Memory Virtualization.....13
1.4.1.1.2.3 Device Virtualization.....15
1.4.1.2 Storage Virtualization.....19
1.4.1.2.1 Overview.....19
1.4.1.2.2 Key Features.....19
1.4.1.2.2.1 Centralized Storage.....19
1.4.1.2.2.2 Distributed Storage.....24
1.4.1.3 Network Virtualization.....26
1.4.1.3.1 Overview.....26
1.4.1.3.2 Key Features.....26
1.4.1.3.2.1 Distributed Switch.....26
1.4.1.3.2.2 Security Group.....27
1.4.1.4 Virtual Resource Management.....30
1.4.1.4.1 VM Management.....30
1.4.1.4.1.1 VM Scheduling Policy.....30
1.4.1.4.1.2 VM Cloning.....34
1.4.1.4.2 Bare Metal Management.....35
1.4.2 Data Protection.....36
1.4.2.1 Snapshot Management.....36
1.4.2.2 Backup and Recovery.....42
1.4.2.2.1 Data Backup.....42
1.4.2.2.1.1 Data Replication.....43
1.4.2.2.1.2 Data Transfer.....43
1.4.2.2.1.3 Data Retention.....45
1.4.2.2.2 Data Recovery.....46
1.4.3 Operations Management.....47
1.4.3.1 Management Node Monitoring.....47

1.4.3.2 Monitoring and Alarms....48
1.5 Reliability....49
1.5.1 Compute High Availability....49
1.5.2 Network High Availability....49
1.5.3 Management Node High Availability....50
1.6 Security....51
1.6.1 Compute Security....51
1.6.1.1 HTTPS-encrypted UI Login....51
1.6.1.2 VM Console....51
1.6.1.3 High Availability....52
1.6.1.4 IP/MAC/ARP Spoofing Protection....53
1.6.1.5 Images and Snapshots....53
1.6.1.6 Encrypted Password Storage....54
1.6.1.7 Resource Deletion Protection....54
1.6.1.8 Monitoring and Alarms....55
1.6.2 Network Security....56
1.6.2.1 Security Group....56
1.6.3 Access Control Security....56
1.6.3.1 Two-Factor Authentication....56
1.6.3.2 AccessKey Authentication....56
1.6.3.3 Tasks and Events....56
1.7 Openness and Compatibility....57
1.7.1 Open APIs....57
1.7.2 Hardware Compatibility....57

Glossary....58

1 Technical Whitepaper

1.1 Overview

Industry Background

Virtualization is widely used in enterprise data centers. As IT infrastructure expands, management complexity increases. Virtualization allows infrastructure resources to be scheduled and allocated dynamically, addressing limitations of traditional architectures such as low resource utilization, data security risks, complex maintenance, and long service deployment cycles. As more performance-intensive applications are virtualized, platform selection must also consider performance, security, reliability, and compatibility.

Product Overview

NexaSphere is designed around four core principles: high performance, strong security, high reliability, and technological independence. It combines the ZStack Cloud engine and 4S capabilities with server, network, and storage virtualization technologies and intelligent operations features to provide a next-generation server virtualization platform. You can use NexaSphere to build a virtualized data center quickly, or combine it with the broader Nexavm Technologies AG product portfolio to deliver integrated solutions from IaaS to PaaS.

Applications

High-Performance Workloads and Database Virtualization:

As the core of modern applications, databases are responsible for storing, managing, and processing large volumes of data. Their performance and reliability are critical to application stability. NexaSphere provides resource isolation and scalability for more efficient use of database server hardware. CPU, memory, and storage resources can be allocated and adjusted dynamically for different application requirements. In addition, NexaSphere delivers flexible backup and recovery capabilities, along with high availability and fault tolerance.

Intelligent Manufacturing:

Manufacturers are increasingly using application innovation and intelligent manufacturing to drive digital transformation and sustainable growth. NexaSphere provides a robust digital foundation for systems such as DMP, ERP, and OA. By combining NexaSphere with next-generation AI and advanced manufacturing technologies, organizations can implement predictive management, digital operations, and data-driven product lifecycle management in scenarios such as:

- For production testing, use NexaSphere to simulate production environments and processes, predict product performance, optimize workflows, and reduce physical testing and trial-and-error costs.
- For supply chain management, use NexaSphere to share supply chain information, forecast demand in real time, and schedule resources dynamically.
- For equipment maintenance, use NexaSphere virtualization to represent physical devices as virtual instances for remote monitoring, fault diagnosis, and maintenance.
- For training and education, use NexaSphere to create virtual training environments that simulate production scenarios and operating procedures.

Production Environment Hosting:

Traditional production environments often face challenges such as complex configuration and deployment, environmental conflicts, physical resource constraints, data security risks, and difficult troubleshooting. NexaSphere effectively addresses these issues through the following capabilities:

- **Rapid Deployment and Environment Isolation**: NexaSphere enables quick provisioning of virtual machines, streamlining environment setup and configuration. It allows the creation of multiple isolated virtual environments tailored to different development and testing projects, preventing inter-project interference and improving R&D efficiency.
- **Efficient Utilization of Physical Resources**: By pooling and sharing physical server resources through virtualization, NexaSphere allows multiple virtual machines to run on a single physical server. This improves resource utilization and reduces hardware costs. It also supports dynamic resource allocation to adapt to changing business needs.
- **Business Data Protection**: NexaSphere supports on-demand VM snapshots and rollback, protecting business data during changes and recovery operations.
- **Flexible Development Environments**: NexaSphere provides adaptable development environments for business teams. Each developer can work within their own VM for development and testing, independently managing and configuring their environment to avoid conflicts and enhance both productivity and output quality.
- **Fault Isolation and Fault Tolerance**: NexaSphere ensures that if one virtual machine fails, others continue operating normally. This reduces downtime caused by software or hardware failures and enhances business continuity and stability.

Virtualization Infrastructure Modernization:

NexaSphere combines compute, storage, and network virtualization with unified management and elastic resource scaling, helping organizations build simple, stable, secure, and efficient IT infrastructure.

1.2 Technical Advantages

High Performance

High-performance workloads place increasing demands on the underlying NexaSphere infrastructure. To meet these requirements, NexaSphere provides the following advantages:

- Exceptional high-concurrency capability, significantly accelerating service deployment
- Minimized computational resource overhead
- Efficient utilization of diverse network resources
- Flexible integration with high-performance storage solutions

High Security

With the continuous improvement of network infrastructure and the increasing migration of data to online environments, ensuring the secure and stable operation of services has become a core priority. NexaSphere inherently provides comprehensive, multi-layered security capabilities:

- Network traffic control: Distributed firewalls and security groups filter north-south and east-west traffic to protect workloads from network threats.
- Data protection: Commercial cryptography support helps organizations establish a strong cryptographic foundation for network and data security.
- Workload protection: A kernel-level, agentless virtualization security engine detects and removes threats to protect business data.

High Reliability

As a core infrastructure platform, NexaSphere provides the foundation for business workloads and data center operations. A reliable NexaSphere environment provides the following advantages:

- Automatic recovery after power failure
- Zero data loss
- Continuous availability in daily operations

Technological Independence

Control over core virtualization technologies helps organizations maintain a secure and sustainable infrastructure foundation. With its independently developed architecture, NexaSphere provides the following advantages:

- Broad compatibility with open infrastructure ecosystems
- Open interfaces for external integrations
- Self-service download, deployment, and evaluation

1.3 Product Architecture

1.3.1 Software Architecture

NexaSphere software architecture highlights:

- Fully Asynchronous Architecture: Asynchronous messaging, methods, and HTTP calls.
  - Asynchronous messaging: Services communicate via a message bus. When calling a service, the source service sends a message to the destination service, registers a callback function, and returns immediately. Upon task completion, the destination service triggers the callback with results. Asynchronous messages can be processed in parallel.
  - Asynchronous methods: Services communicate through asynchronous messages. Components or plugins inside each service use asynchronous method calls, following the same pattern as asynchronous messaging.
  - Asynchronous HTTP calls: Each plugin has an agent that puts a callback URL in the HTTP header of every request. After task completion, the agent sends the response to the caller's URL.
  - Based on these three asynchronous approaches, NexaSphere builds a layered architecture to ensure all components operate asynchronously.
  - With this fully asynchronous architecture, a single management node can process tens of thousands of concurrent API requests per second, while managing tens of thousands of hosts and hundreds of thousands of virtual machines.
- Stateless Services: Each request is independent.
  - Compute node agents, storage agents, network services, console proxy services, and configuration services have no dependencies between requests. Each request carries all required context, so nodes do not need to retain request state.

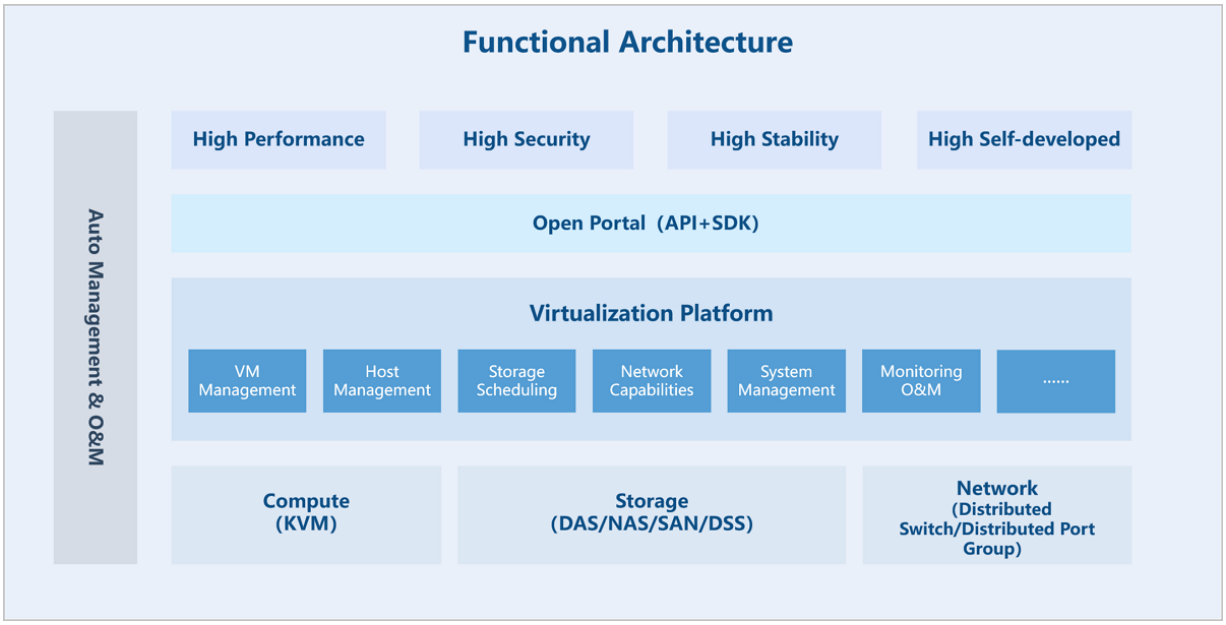
  - Resources such as management nodes and compute nodes are identified by UUIDs and assigned through a consistent-hashing ring. Message senders do not need to know which service instance handles a resource, and service instances process messages without exchanging resource state.
  - Little information is shared among management nodes. Therefore, a minimum of two management nodes can meet the requirements of high availability and scalability.
  - Stateless services improve resilience because restarting a server does not lose request state. They also simplify scaling management services out or in.
- Lock-Free Architecture: Consistent hashing algorithm.
  - The consistent hashing algorithm ensures all messages of the same resource are always processed by the same service instance. This message aggregation to specific nodes reduces synchronization and parallel processing complexity.
  - Work queues replace lock contention. Operations that must be serialized for the same resource are queued in memory, while operations on different resources can run in parallel.
  - This queue-based, lock-free architecture increases task concurrency and system performance.
- In-Process Microservices: Decoupled microservices.
  - A message bus isolates and controls services, such as virtual machine services, identity authentication services, snapshot services, volume services, network services, and storage services. All microservices are enclosed in the management node process. These services communicate with each other through the message bus. All messages are first sent to the message bus, then forwarded to the destination service using consistent hashing ring.
  - In-process microservices use a star topology in which each service runs independently. This design decouples centralized control functions and isolates services so that a service failure does not directly affect other components.
- Versatile Plugin System: Horizontally scalable plugins.
  - Every plugin provides services independently. Any newly added plugin has no impact on other existing plugins.
  - Plugins support the Strategy and Observer patterns. Strategy plugins implement parent-class interfaces for specific functions. Observer plugins register listeners for internal events and trigger the corresponding workflow when an event occurs.
  - This horizontally extensible plugin system allows NexaSphere to add and update capabilities without weakening the core architecture.

- Workflow Engine: Sequence-based management and error rollback.
  - Each workflow is defined in XML. If an error occurs, the workflow rolls back the completed steps in reverse order and cleans up temporary resources.
  - A workflow can contain subworkflows that implement additional business logic.
- Tag System: Enables business logic change and resource attributes.
  - Extend or modify business logic dynamically through system tags and plugins.
  - Group and categorize resources with tags, enabling resource searches by specific tags.
- Cascade Framework: Supports chained resource operations.
  - The Cascade framework manages dependencies through chained resource operations. For example, detaching or deleting a resource triggers the corresponding operations on related resources.
  - Resources can be added to the Cascade framework as plugins. Their addition or removal does not affect other resources.
  - The cascading mechanism enables flexible and lightweight resource configuration, allowing rapid adaptation to customer configuration changes.
- Automated Deployment: Agentless auto-deployment with Ansible.
  - Agentless Ansible automates dependency installation, physical resource configuration, and agent deployment. The process requires no additional user intervention, and reconnecting an agent can trigger its upgrade.
- Comprehensive Query APIs: Access any attribute of any resource.
  - Provides comprehensive query APIs that support large numbers of conditions and flexible condition combinations.

1.3.2 Functional Architecture

NexaSphere uses the functional architecture illustrated in *Figure 1-1: Functional Architecture*.

Figure 1-1: Functional Architecture



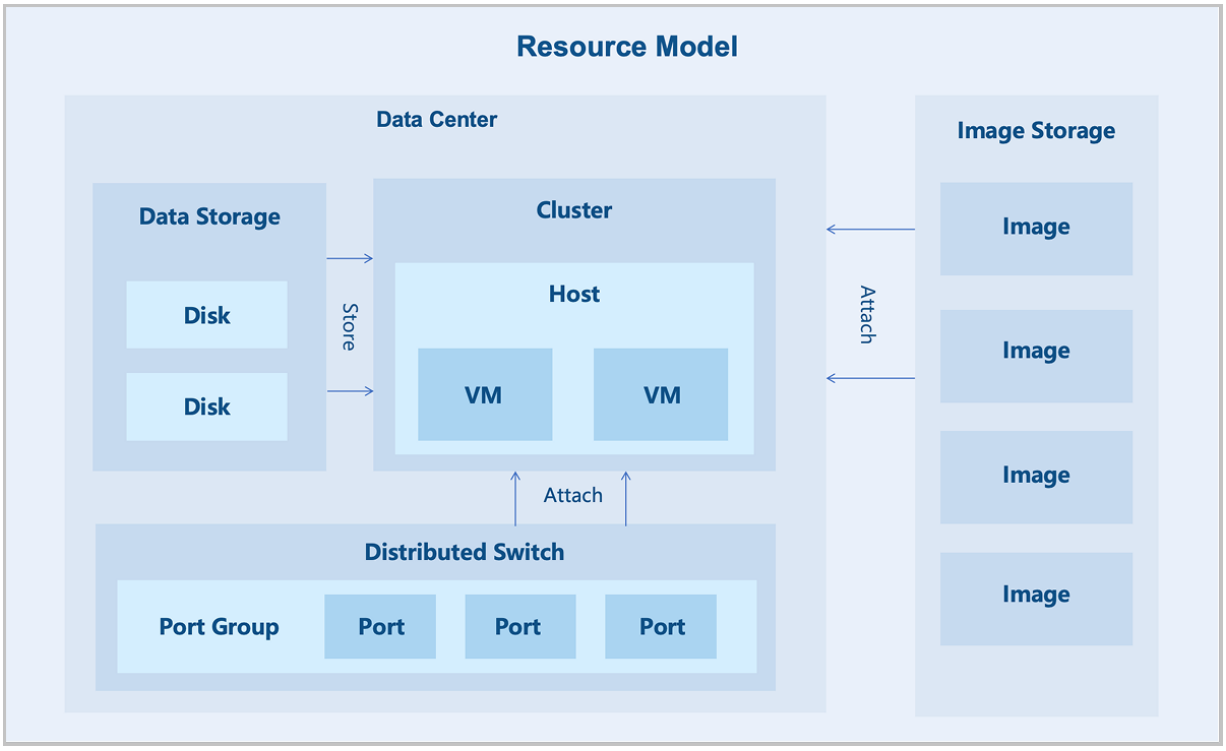
NexaSphere functional architecture highlights:

- Provides enterprise-grade data center infrastructure services for compute, storage, and network resource management. The underlying platform uses KVM virtualization and supports local, NFS, SAN, and distributed storage, together with distributed switches and distributed port groups.
- With NexaSphere as its core engine, the platform uses a message bus to coordinate the MariaDB database and service modules. Core functions include VM management, host management, storage scheduling, network services, system administration, monitoring, and auditing.
- Provides Java and Python SDKs and RESTful APIs for resource scheduling and management.

1.3.3 Resource Model

NexaSphere uses the resource model illustrated in *Figure 1-2: Resource Model*.

Figure 1-2: Resource Model



NexaSphere has the following main resources:

- Data Center: The largest resource namespace in the virtualization platform. It contains clusters , hosts, data storage, distributed switches, distributed port groups, and other resources.
- Cluster: A logical collection of hosts, or compute nodes.
- Host: An x86 or ARM physical server that runs the KVM hypervisor and provides compute, network, and storage resources to VMs.
- VM: A virtual server that runs on a physical host and can run an operating system and business applications like a physical server.
- Data Storage: A virtualized resource that provides storage space for VMs and application data. Data storage can use local or shared network storage.
- Distributed Switch: A virtual switching device that provides unified virtual network management and monitoring for VMs in a cluster.
- Distributed Port Group: A logical group of distributed switch ports used to manage port configuration.
- Image Storage: A virtualized resource that stores image files used by VMs or disks. It can be a standalone or distributed image storage.
- Image: An image file used by a VM or disk. Images include system images and disk images.

NexaSphere resources maintain the following two types of relationships:

- Hierarchical relationships: Describe parent-child, sibling, grandparent-grandchild, and peer relationships between resources.

  Definitions:

  - Parent-Child: Resource A is the parent or child of Resource B. For example, a host belongs to a cluster, and a VM runs on a host.
  - Sibling: Resource A and Resource B share the same parent. For example, clusters and distributed switches, or clusters and data storage. Both pairs share the data center as their parent.
  - Grandparent-Grandchild: Resource A is the direct grandparent or grandchild of Resource B . For example, a data center is the parent of a cluster, a cluster is the parent of a host, and a host is the parent of a virtual machine. Therefore, the data center is the grandparent of the host, and the cluster is the grandparent of the virtual machine.
  - Peer: Resource A and Resource B do not have any of the preceding relationships but collaborate in specific scenarios. For example, data storage and image storage work together to provide services to clusters.
- Cardinality relationships: Describe 1:n (one-to-many), n:1 (many-to-one), and n:n (many-to-many) relationships.

  Definitions:

  - **1:n**: Resource A can contain, add, or be associated with multiple Resource B instances. For example, one cluster can contain multiple hosts, and one distributed switch can be associated with multiple clusters.
  - **n:1**: Multiple Resource A instances can belong to or be associated with one Resource B instance. For example, multiple hosts can belong to the same cluster, and multiple clusters can be associated with the same distributed switch.
  - **n:n**: Resource A can be associated with multiple Resource B instances, and Resource B can also be associated with multiple Resource A instances. For example, one image storage can be associated with multiple data centers, and one data center can use multiple image storage resources.

1.4 Core Design

1.4.1 Resource Virtualization

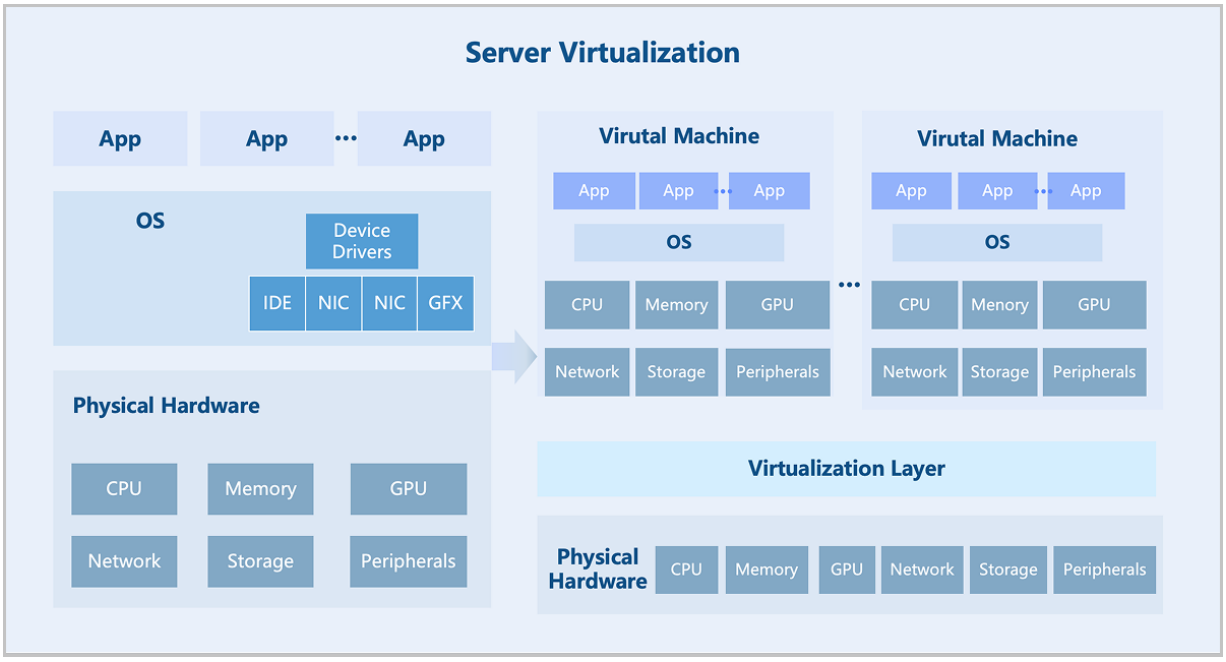
1.4.1.1 Compute Virtualization

1.4.1.1.1 Overview

Compute virtualization abstracts physical server resources into logical resources through virtualization technology, enabling a single physical server to function as multiple isolated virtual machines. It pools hardware resources such as CPU, memory, disk, and I/O devices into a virtual resource pool for unified dynamic management. Consequently, it improves resource utilization, reduces system management costs, and increases IT agility to adapt to business changes.

See *Figure 1-3: Server Virtualization* for an illustration.

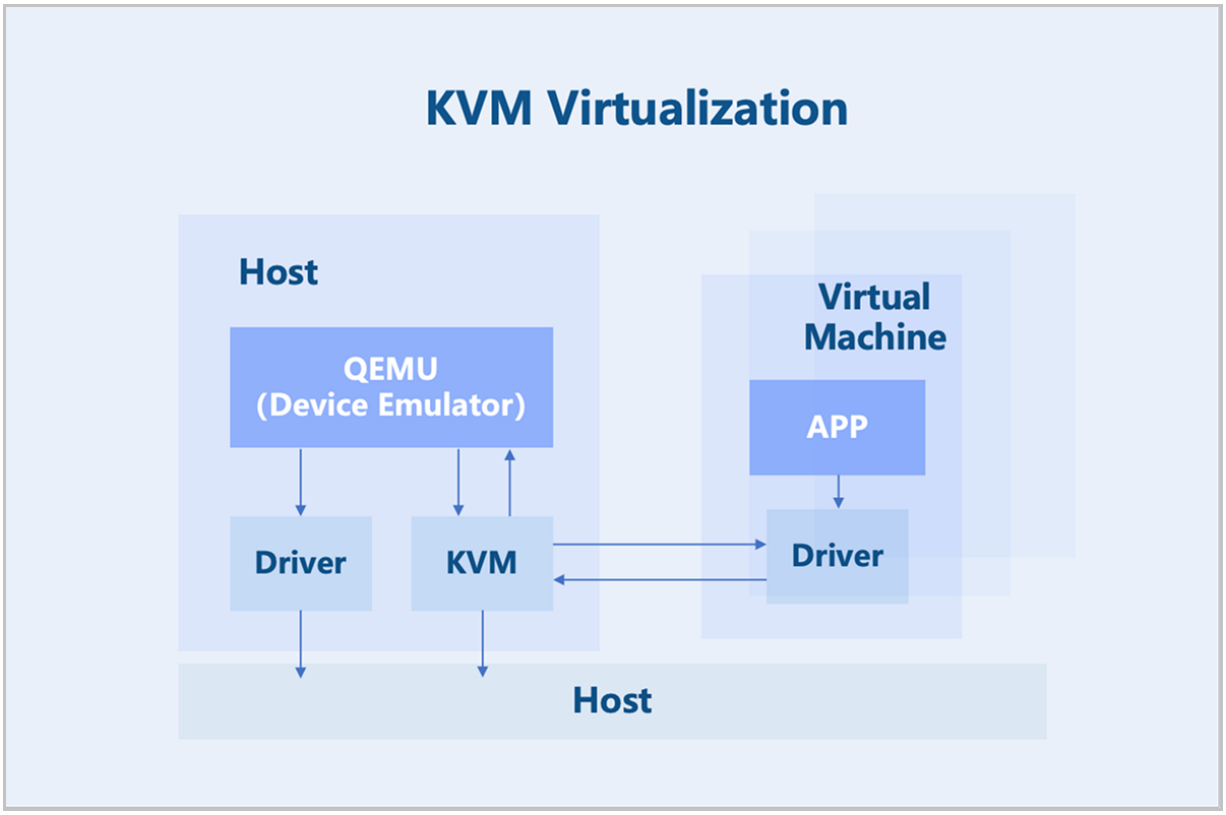
Figure 1-3: Server Virtualization



NexaSphere uses KVM-based hardware virtualization. KVM is a Linux kernel module that turns the Linux kernel into a hypervisor. Each VM runs as a Linux process scheduled by the standard Linux scheduler, allowing KVM to use existing kernel capabilities such as memory management and CPU scheduling. KVM provides CPU and memory virtualization, while QEMU provides the user-space device models required for I/O device virtualization and manages the virtual devices presented to VMs.

See *Figure 1-4: KVM Virtualization* for an illustration.

Figure 1-4: KVM Virtualization



1.4.1.1.2 Key Features

1.4.1.1.2.1 CPU Virtualization

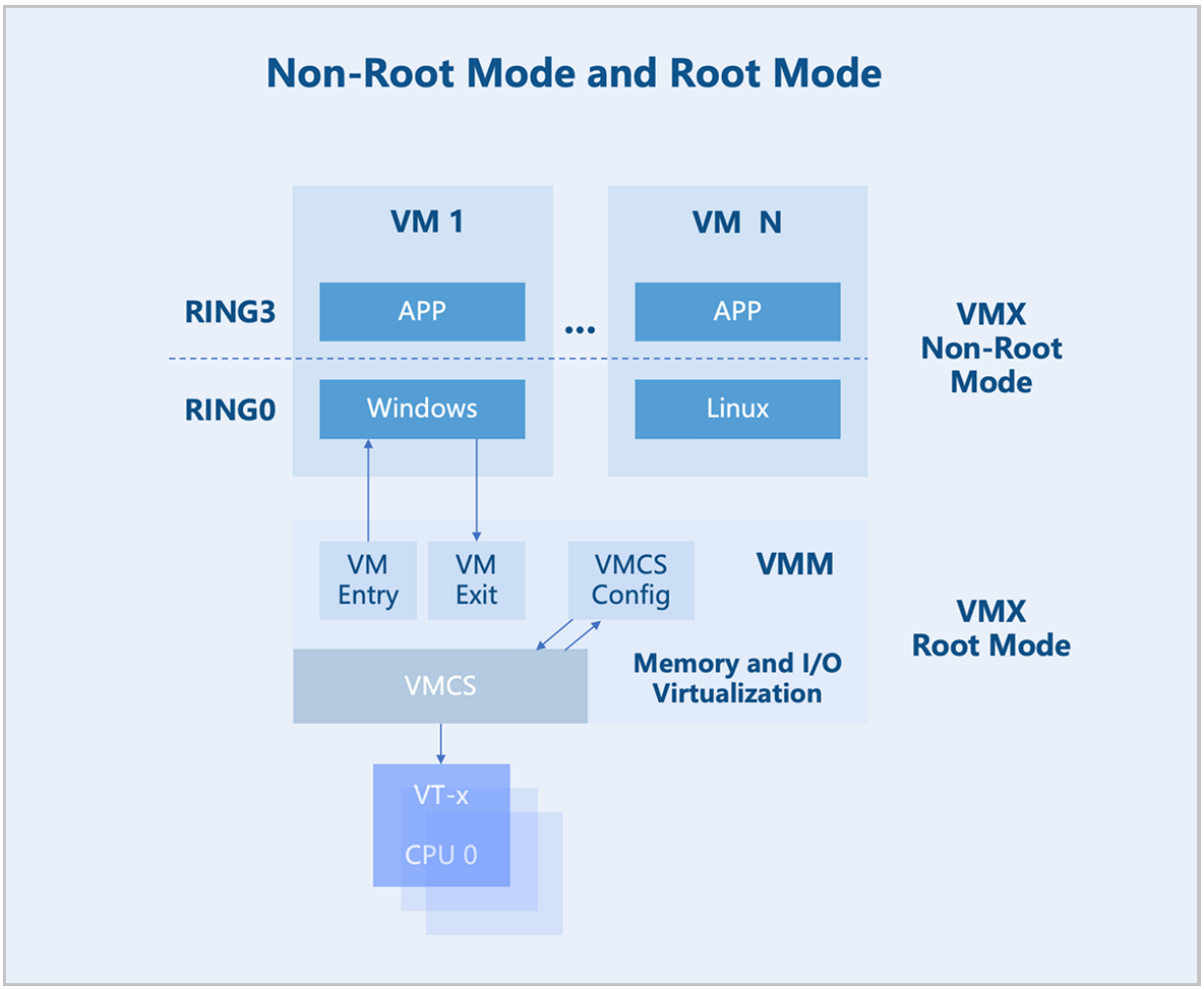
In the x86 architecture, CPUs generally have four privilege levels (RING0 to RING3) for operating systems and applications to access hardware. Linux uses only two of these levels: RING0 (kernel mode) and RING3 (user mode).

VMX Root Mode and VMX Non-Root Mode: For hardware-assisted virtualization, CPUs introduce two operation modes to enable virtual machines to run without modifying the operating system: VMX root mode and VMX non-root mode. The host runs in root mode, with its kernel in RING0 and user-mode programs in RING3. The virtual machine runs in non-root mode, with its kernel in RING0 and user-mode programs in RING3.

VM Exit and VM Entry: A switch from non-root mode to root mode occurs when a VM in non-root mode encounters an external interrupt or page fault, or executes the VMCALL instruction to invoke VMM services. This transition is called VM Exit. Conversely, when the VMM executes the VMLAUNCH or VMRESUME instruction to switch to non-root mode, the hardware loads the VM context and resumes VM instruction execution. This transition is called VM Entry.

See *Figure 1-5: Non-Root Mode and Root Mode* for an illustration.

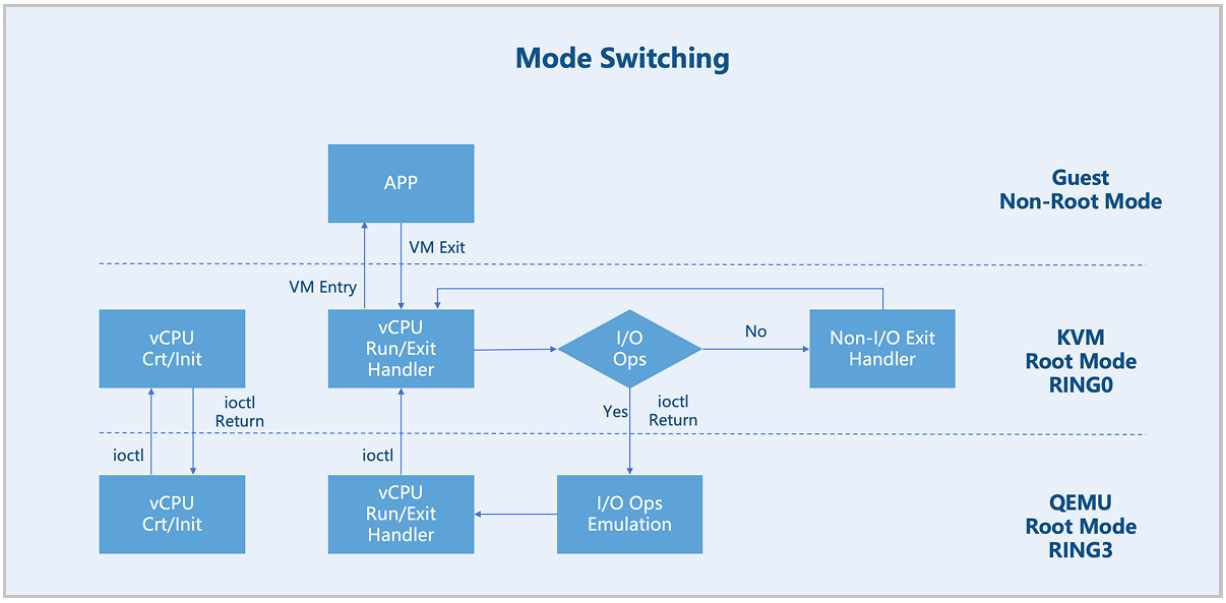
Figure 1-5: Non-Root Mode and Root Mode



After a virtual machine triggers a VM Exit from non-root mode to root mode, KVM takes further action based on the exit reason. If the exit is due to an I/O operation, KVM delegates the processing to QEMU. For non-I/O exits, KVM handles them directly. After processing, KVM initiates a VM Entry to switch back to non-root mode for virtual machine execution.

See *Figure 1-6: Mode Switching* for an illustration.

Figure 1-6: Mode Switching



1.4.1.1.2.2 Memory Virtualization

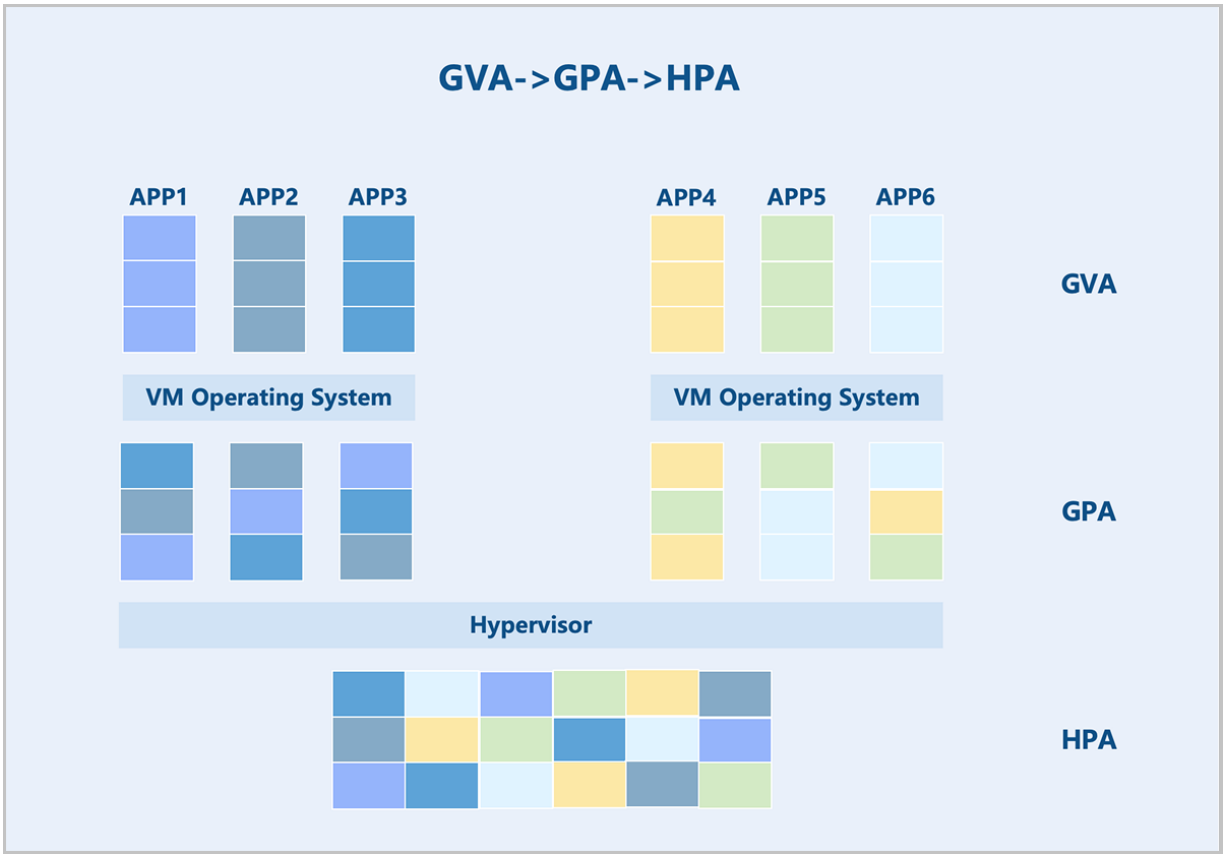
The Virtual Machine Monitor (VMM) manages and allocates physical memory for each virtual machine. The guest OS sees a virtualized Guest Physical Address (GPA) space. The OS memory management module maps Guest Virtual Addresses (GVAs) to GPAs. The target address in instructions is also a GPA. In a non-virtualized environment, such an address would be the actual physical address. However, in a virtualized scenario, this address cannot be used directly. Instead, the VMM must first translate the GPA into a Host Physical Address (HPA), which is then executed by the physical processor.

The introduction of the GPA space requires memory virtualization to address two main issues:

- Maintaining the mapping relationship between GPAs and HPAs.
- Translating GPAs into HPAs whenever a virtual machine accesses a GPA based on the established mapping.

See *Figure 1-7: GVA->GPA->HPA* for an illustration.

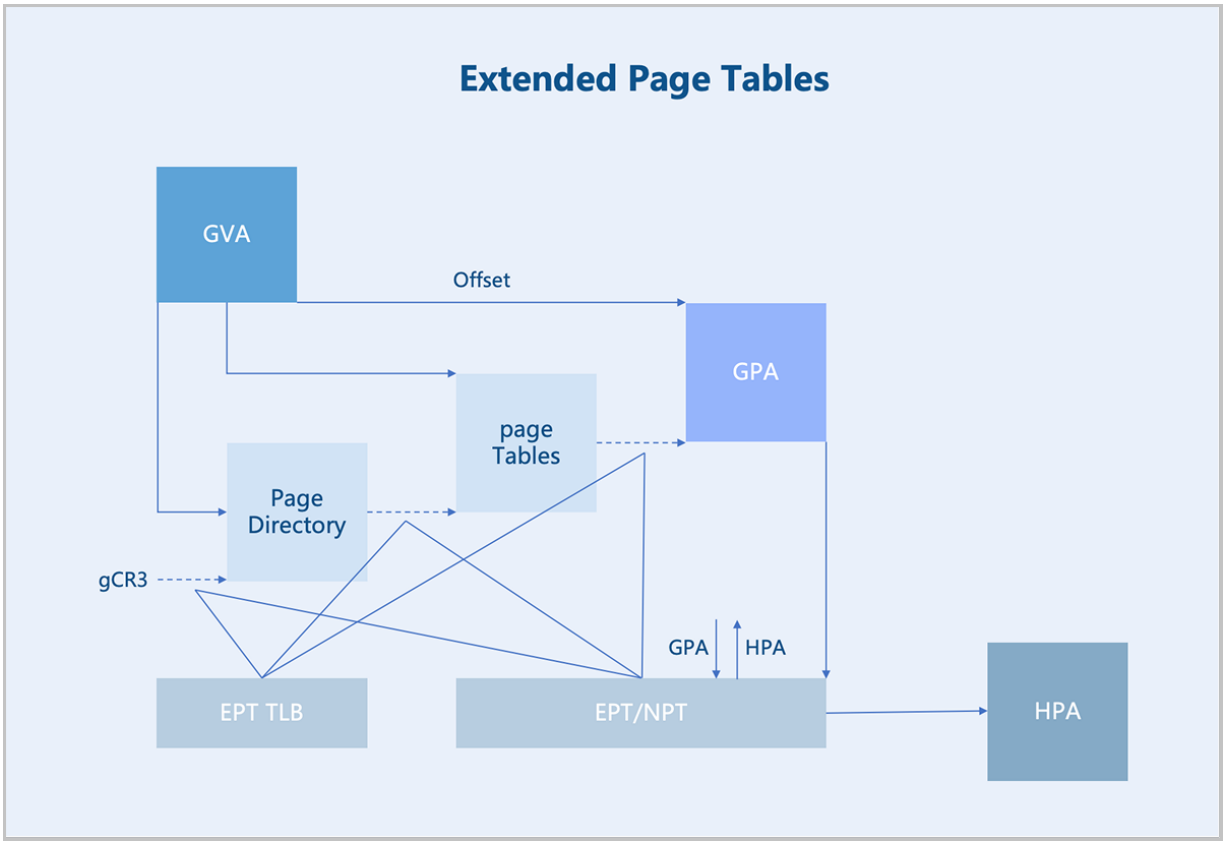
Figure 1-7: GVA->GPA->HPA



Hardware-assisted memory virtualization uses Extended Page Tables (EPT) technology to translate GVAs into HPAs at the hardware level.

See *Figure 1-8: EPT* for an illustration.

Figure 1-8: EPT



1.4.1.1.2.3 Device Virtualization

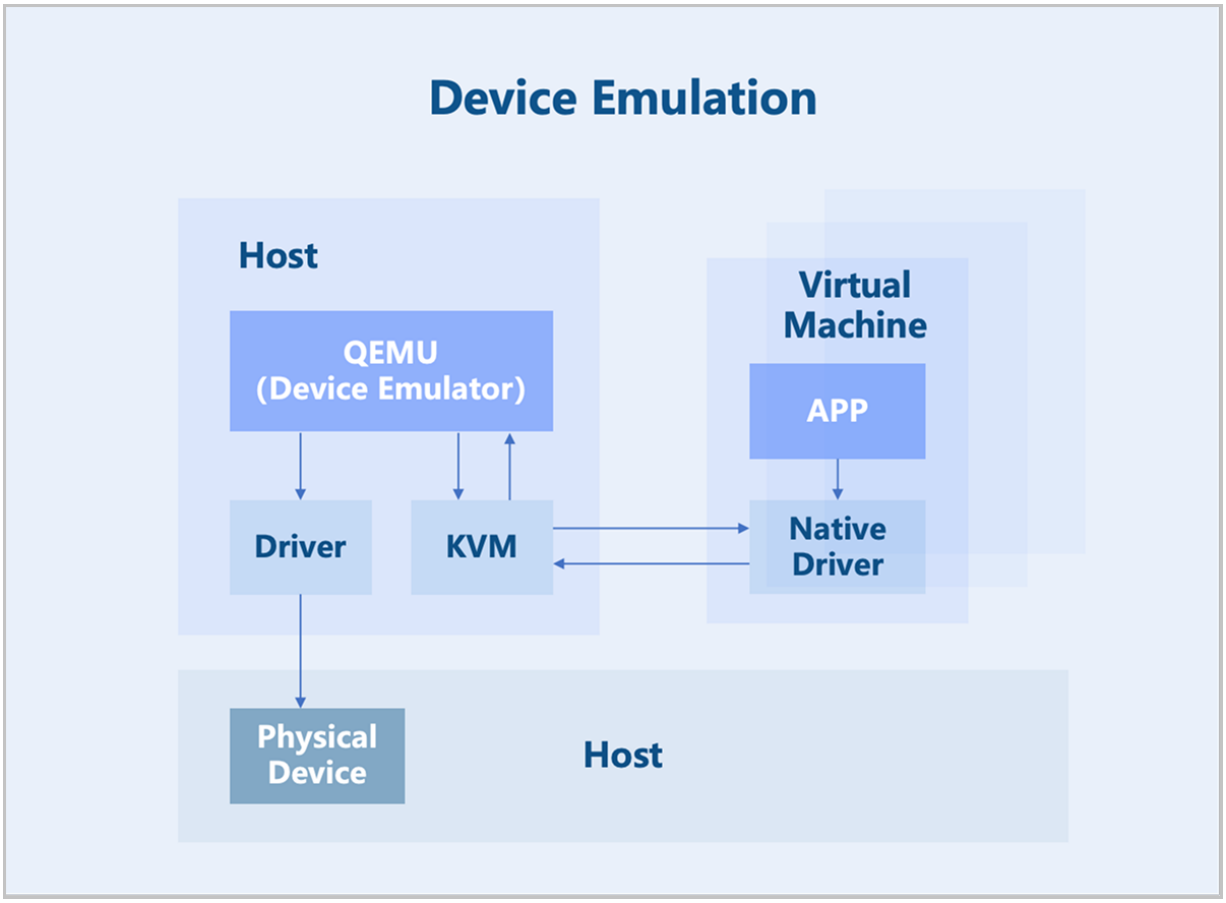
There are three main approaches to device virtualization: device emulation, paravirtualized devices, and device passthrough.

Device Emulation

Device emulation uses QEMU device models to emulate interfaces presented by physical devices . The guest operating system can therefore use its native drivers. Emulated devices provide broad compatibility but support only the functions implemented by their software models and generally have lower performance than paravirtualized or passthrough devices.

See *Figure 1-9: Device Emulation* for an illustration.

Figure 1-9: Device Emulation



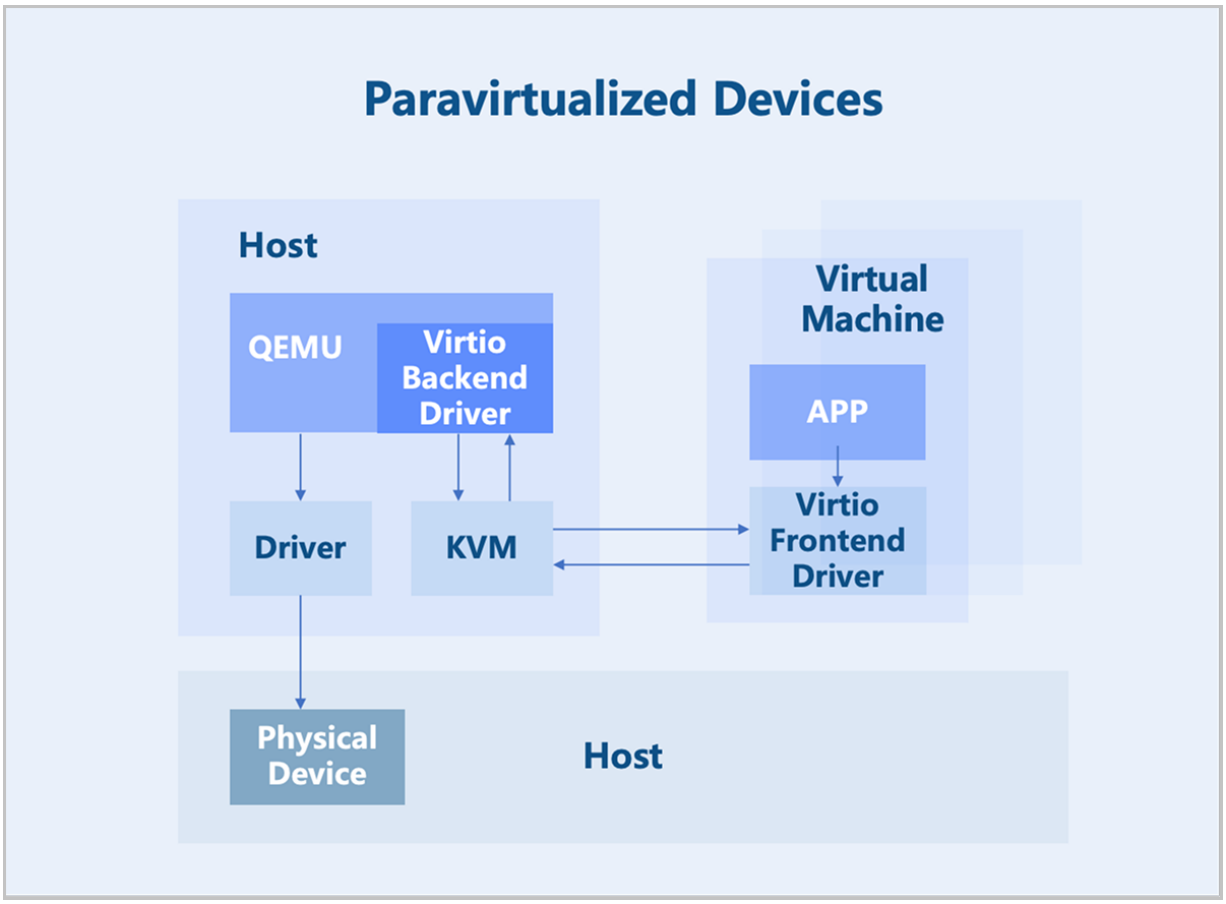
Paravirtualized Devices

Paravirtualized devices implement frontend and backend drivers. Utilizing a frontend driver in the virtual machine, requests are directly sent to a backend driver on the host through a transactio n-based communication mechanism, significantly reducing context switching overhead and improving performance over full device emulation. However, since the Virtio backend driver is still implemented in QEMU, the I/O processing path involves multiple switches between user space and kernel space. To further enhance performance, the functionality of the Virtio backend driver can be moved into the kernel space. This implementation is known as the Vhost-kernel backend. With this change, data transmission requires only a single switch from user space to kernel space , thereby improving performance.

As technology evolves, moving data processing to user space provides greater flexibility. Consequently, modifications to the original Vhost architecture led to the Vhost-user backend, which works with relevant user-space libraries from DPDK and SPDK to further boost performanc e.

See *Figure 1-10: Paravirtualized Devices* for an illustration.

Figure 1-10: Paravirtualized Devices

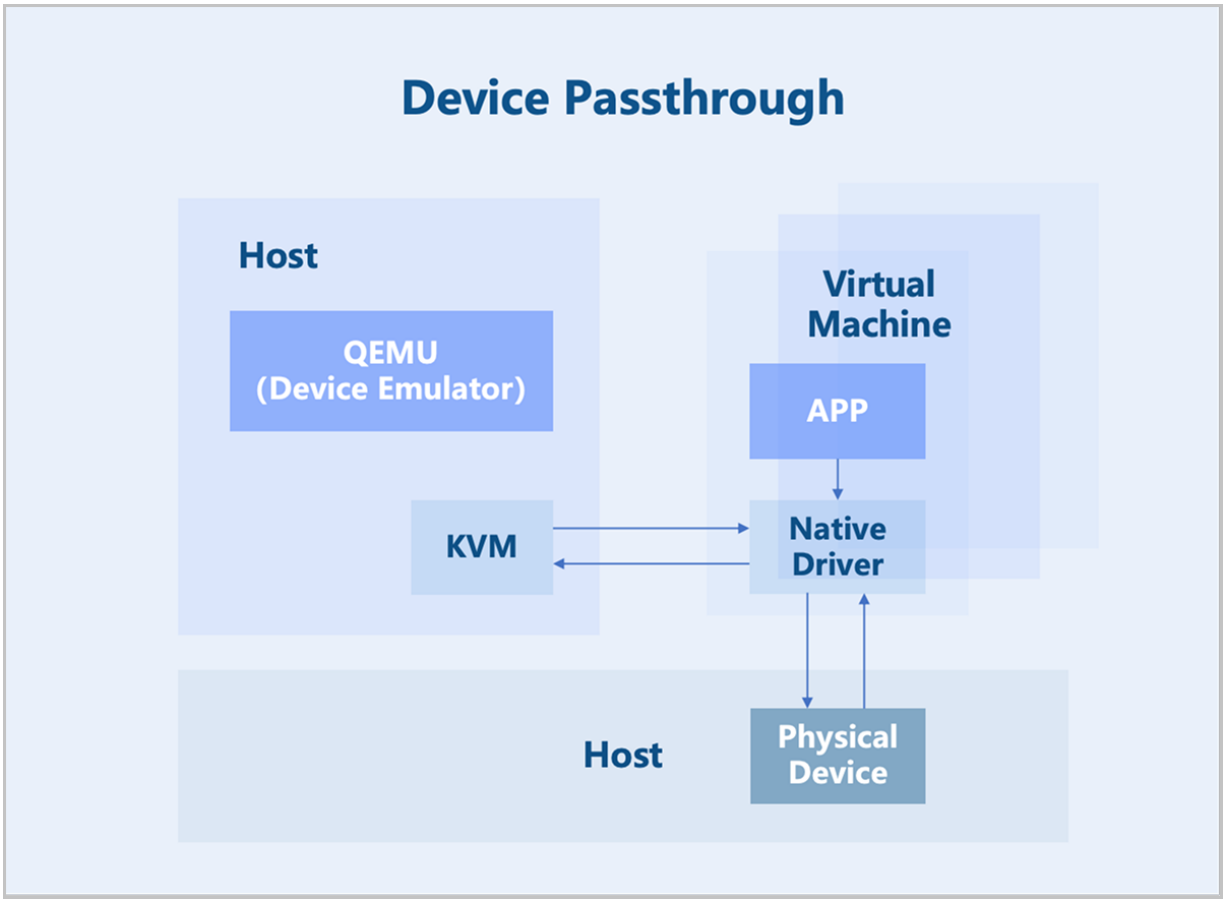


Device Passthrough

Device passthrough uses hardware-assisted I/O virtualization to map a physical PCI or PCIe device directly into a VM address space. The VM operates the device through its native driver and can achieve near-native performance. A passed-through device is dedicated to one VM and cannot be shared with other VMs.

See *Figure 1-11: Device Passthrough* for an illustration.

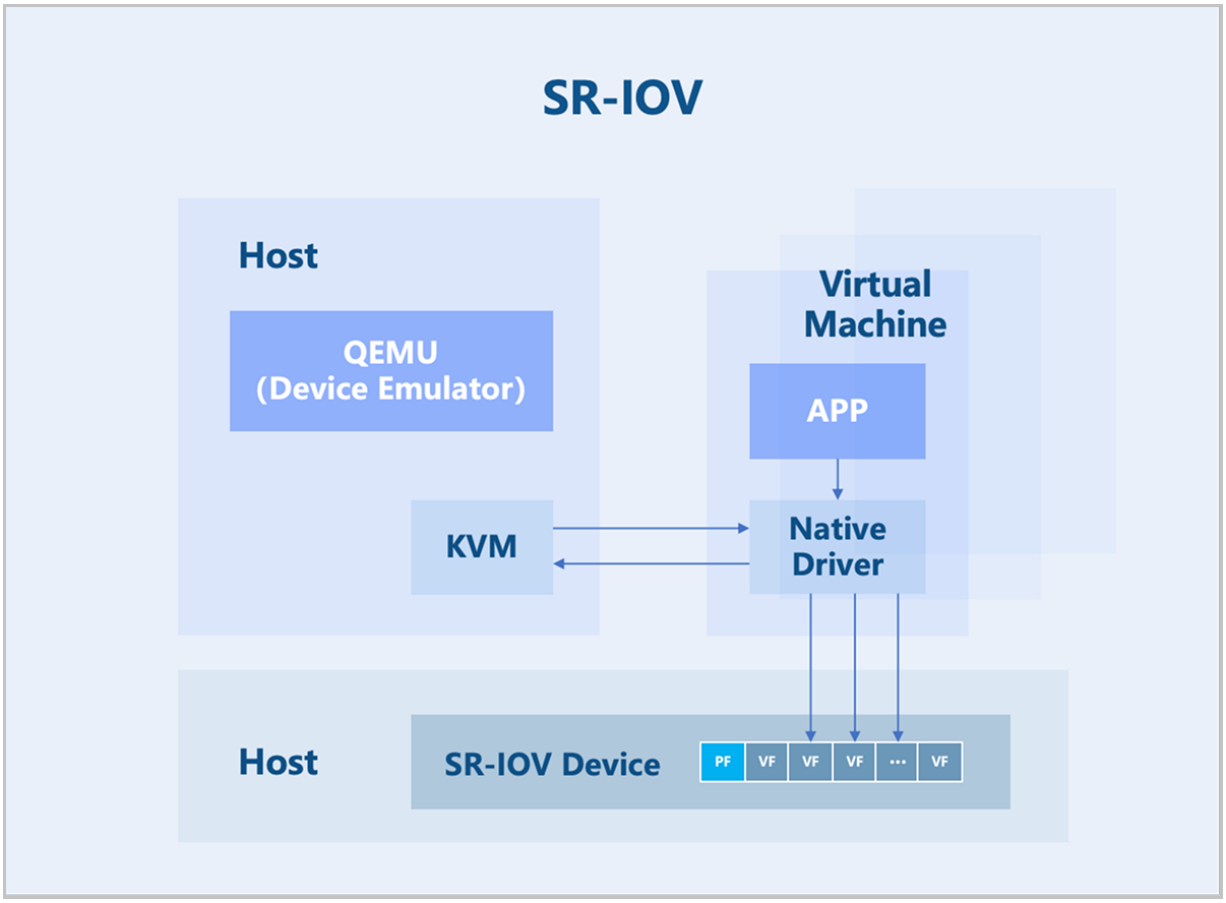
Figure 1-11: Device Passthrough



SR-IOV is an extension to the PCIe specification defined by the PCI-SIG. Its purpose is to provide a standardized specification for granting virtual machines independent memory spaces, interrupts, and DMA data streams. SR-IOV enables a single physical PCIe device (Physical Function, PF) to be virtualized into multiple virtual PCIe devices (Virtual Functions, VFs). These VFs can then be passed through directly to virtual machines using device passthrough technology. This allows a single physical PCIe device to support multiple virtual machines.

See *Figure 1-12: SR-IOV* for an illustration.

Figure 1-12: SR-IOV



1.4.1.2 Storage Virtualization

1.4.1.2.1 Overview

Storage virtualization pools storage resources from servers for unified management and allocation. It exposes storage interfaces to upper-layer services, allowing workload VMs to obtain storage capacity from the resource pool as needed. NexaSphere supports both centralized and distributed storage systems.

1.4.1.2.2 Key Features

1.4.1.2.2.1 Centralized Storage

Centralized storage stores data on a central system consisting of one or more servers. Services are deployed centrally, and data is managed through the storage controllers. Centralized storage includes DAS, NAS, and SAN. You can select a storage type according to workload and data storage requirements.

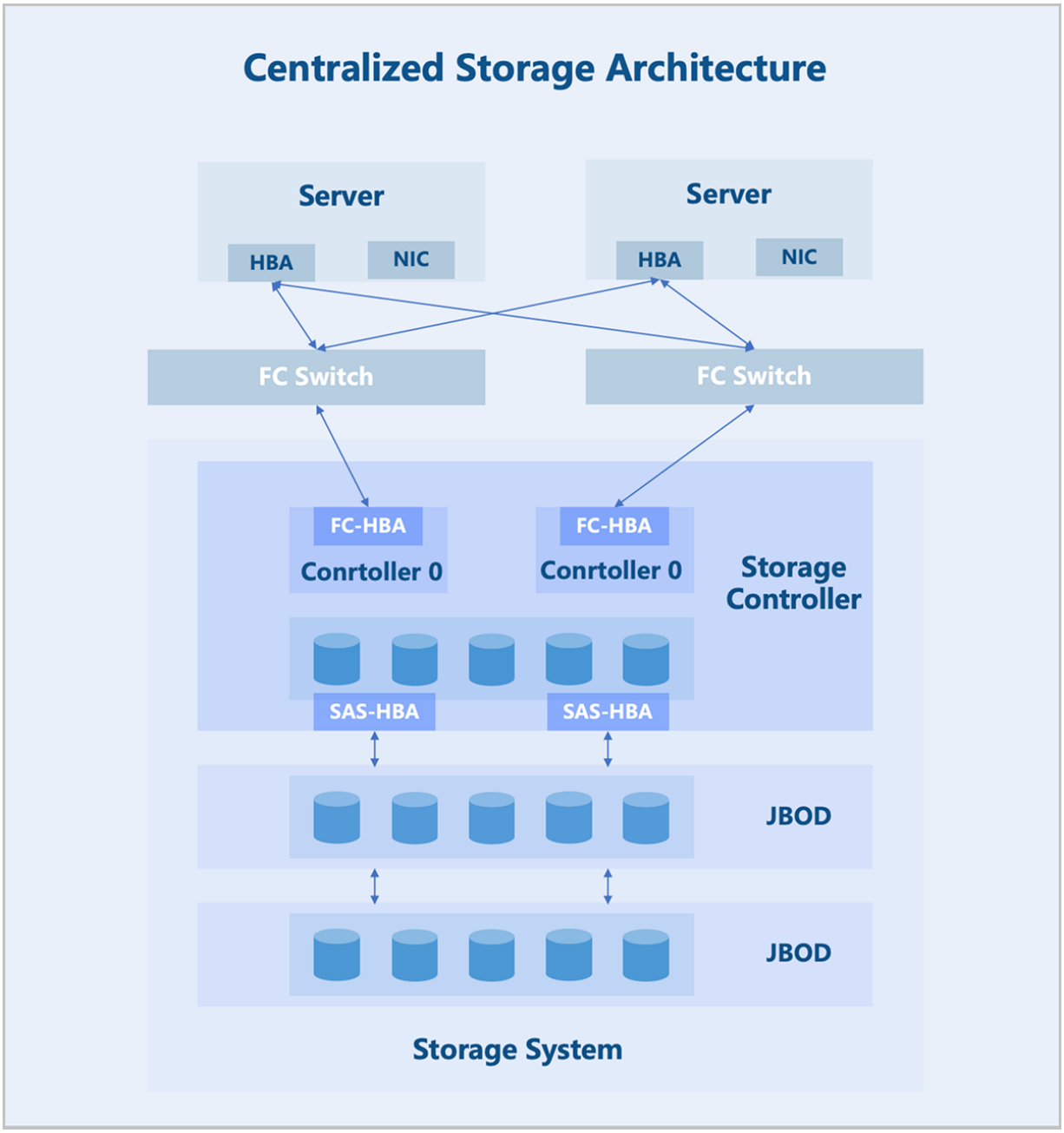
Functional advantages:

- High Performance and High Reliability: Centralized storage keeps data in a centrally managed system, reducing the risks associated with fragmented storage and improving storage performance and reliability.
- Scalability: Centralized storage can expand over the network as data volume and service requirements grow.
- Security: Access control, data encryption, and other security measures protect data confidentiality and integrity.

The storage controller is the core component of a centralized storage architecture. A storage system typically has two controllers that provide redundancy, preventing a single controller failure from making the entire system unavailable. Frontend ports provide storage services to servers, while backend ports connect additional storage devices to expand system capacity and form a larger storage pool.

See *Figure 1-13: Centralized Storage Architecture* for an illustration.

Figure 1-13: Centralized Storage Architecture



SAN

NexaSphere supports various centralized storage systems. With SAN storage, LUNs allocated on the storage system can be used directly as storage pools for workload VMs. Compared with file system-based data storage, SAN storage is straightforward to deploy, scales flexibly, and can deliver the performance of the underlying physical disks. Supported shared-access protocols include iSCSI, FC, and NVMe-oF.

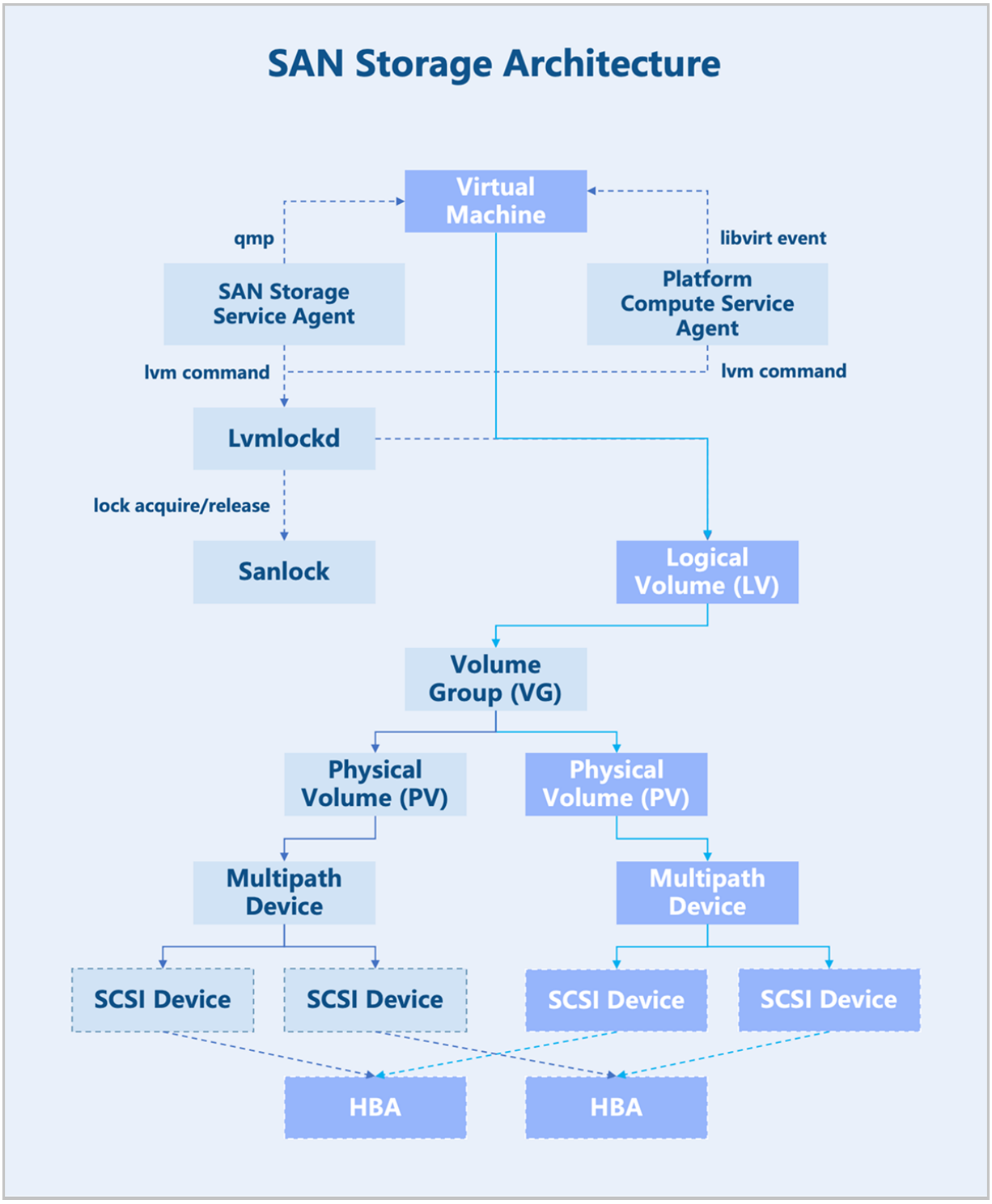
Working principles:

- Block-Level Access: SAN divides data into blocks, each with a unique identifier. Servers read from or write to specific data blocks using these identifiers.
- Parallel Access: Multiple nodes can simultaneously access different blocks within SAN, enabling parallel read and write operations.
- Shared Access: Multiple nodes can share the same SAN, allowing concurrent access to the same data blocks. This is crucial for scenarios requiring data sharing in distributed systems.
- Fault Tolerance: SAN typically provides data redundancy and failure recovery mechanisms to ensure data safety and reliability.
- Scalability: SAN can scale capacity and performance by adding more storage nodes.
- Performance Optimization: SAN usually uses various techniques, such as caching and load balancing, to optimize performance.

In this architecture, a volume on the centralized storage system is mapped to the hosts. Depending on how the storage controllers are connected, the multipath service on each host automatically aggregates SCSI devices with the same WWID into a multipath device. NexaSphere detects multipath devices with the same WWID across all hosts in the cluster and creates a shared volume group (VG) for the selected volume. NexaSphere then creates logical volumes (LVs) from the VG. These LVs correspond to volumes, snapshots, and other resources in NexaSphere. To maintain storage cluster consistency, NexaSphere provides the Sanlock shared-storage locking mechanism for storage heartbeat maintenance, node management, and metadata management.

See *Figure 1-14: SAN Storage Architecture* for an illustration.

Figure 1-14: SAN Storage Architecture



Sanlock + Lvmlockd

Sanlock, a shared-storage lock manager, implements a lease mechanism based on Lamport algorithms such as Delta Paxos and Disk Paxos. The mechanism provides cluster membership

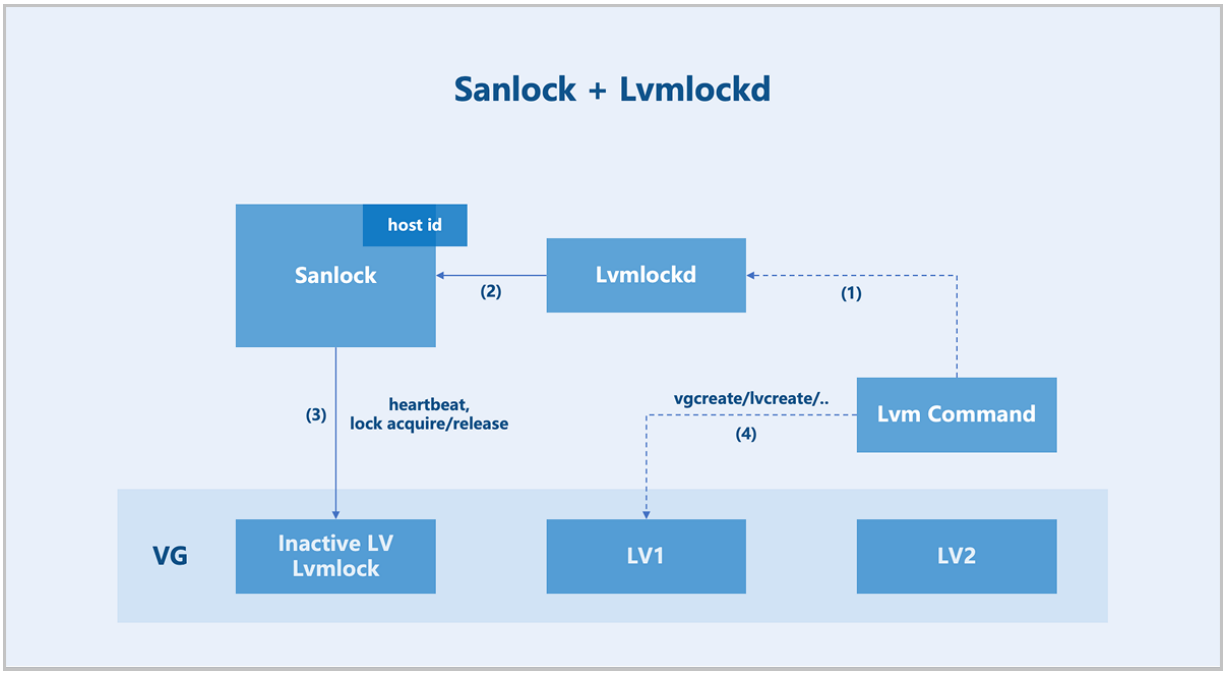
management, heartbeat maintenance, arbitration, and message exchange. Lvmlockd converts these leases into locks that LVM can use.

With Sanlock and Lvmlockd, most LVM commands issued by a server are sent to Lvmlockd first. Lvmlockd translates the LVM operations into lock operations that Sanlock can process.

Different resource operations require different lock permissions. Resource operations proceed only after obtaining the appropriate locks, ensuring the security of metadata and data. This entire process requires no intermediate node coordination or centralized operations, forming a fully distributed architecture.

See *Figure 1-15: Sanlock + Lvmlockd* for an illustration.

Figure 1-15: Sanlock + Lvmlockd



1.4.1.2.2.2 Distributed Storage

NexaSphere supports various distributed storage systems. This section uses nSAN as an example. It manages resources such as servers, physical disks, data disks, block storage volumes, storage pools, and buckets. You can create storage pools with different redundancy schemes and configure storage policies and access permissions for buckets, providing a stable, secure, and cost-effective storage infrastructure.

Functional advantages of nSAN:

- Elastic Scalability: Expands storage capacity and processing capability horizontally by adding nodes to the cluster, accommodating continued data growth without replacing the existing system.
- High Availability: Supports multiple data redundancy schemes, including replication and erasure coding. The replica count of a storage pool can be adjusted online. Because data is distributed across multiple nodes, it remains accessible when one or more nodes fail. Flexible fault-domain policies further maintain storage availability during hardware failures and network interruptions.
- Performance Optimization: Optimizes data processing performance through proprietary cache acceleration technology. It utilizes enterprise-grade NVMe SSDs and SATA SSDs to accelerate read and write operations for backend HDD devices, effectively reducing latency and improving the I/O performance of the storage cluster. Furthermore, it employs load balancing technology to evenly distribute storage I/O loads across all nodes, ensuring optimal utilization of system resources.
- Easy Management: Provides a unified management interface and standard RESTful APIs for physical-resource lifecycle management, maintenance, data operations, and recovery.
- High Cost-Effectiveness: Runs on standard hardware and allows storage resources to scale with service requirements, reducing hardware costs and avoiding overprovisioning.

Architecture of nSAN:

- Hardware Layer: Supports standard x86 servers and integrating existing equipment into a shared pool, helping organizations maximize the value of their current IT assets.
- Platform Layer: Extends consistent hashing with physical deployment and cluster-state information through a fault-domain algorithm. During data migration, the algorithm considers the hardware topology to minimize data movement. For example, it can keep migrations within a server or within the same rack or core-switch domain, reducing the resources consumed by migration. It also distributes data according to cluster state to accommodate different storage devices.
- Data Layer: Uses enterprise-grade NVMe SSDs and SATA SSDs as read and write caches for backend HDDs. Frequently accessed data remains on high-speed SSDs, while an intelligent destaging policy moves cold data to HDDs. This design improves overall cluster performance while providing high-capacity storage for upper-layer services.
- Interface Layer: Provides virtual volume devices to operating systems, NexaSphere, and databases through the RBD interface.

- Service Layer: Integrates with NexaSphere to provide a cloud infrastructure foundation for workloads.

1.4.1.3 Network Virtualization

1.4.1.3.1 Overview

Network virtualization decouples network services from dedicated network hardware. The underlying hardware provides basic packet forwarding, while the virtualization layer provides switching, routing, security groups, and other services with behavior comparable to a physical network.

NexaSphere represents virtual networks through distributed switches and distributed port groups. A distributed switch spans multiple hosts in a data center and provides centralized deployment, management, and monitoring of the virtual network. A distributed port group is a collection of virtual ports on a distributed switch. VMs in the same port group share a consistent network configuration, allowing them to migrate between hosts without changing that configuration. Distributed port groups also provide distributed DHCP services, and DNS can be configured for VMs to support different data center network scenarios.

1.4.1.3.2 Key Features

1.4.1.3.2.1 Distributed Switch

Switches use MAC or IP addresses to forward and control traffic between hosts and between hosts and external networks. In a traditional data center, hosts remain at fixed locations and their connections to switches rarely change, so switch configurations are relatively static.

In virtualized data centers, switches often face new challenges, such as:

- VM MAC and IP addresses cannot always be determined before deployment.
- When a VM migrates, the corresponding configuration on physical switch ports must also be updated, increasing the risk of migration failure.
- When source and destination virtual machines are on the same host, network traffic might bypass the switch, preventing the switch from performing traffic control.

NexaSphere provides distributed switches for forwarding and controlling traffic between VMs and between VMs and external networks.

A distributed switch is a logical switch distributed across each host, consisting of virtual switches, virtual machine NICs, and host NICs.

- The virtual switch forwards network traffic based on MAC addresses.
- Virtual machine NICs connect virtual machines to the virtual switch, providing data channels for virtual machine traffic.
- Host NICs serve as uplink ports for the virtual switch, forwarding cross-host virtual machine traffic to physical switches.

1.4.1.3.2.2 Security Group

In traditional data centers, networks divide into trusted zones, DMZ zones, and untrusted zones. Perimeter firewalls enforce traffic control to ensure network security. In virtualized data centers, perimeter firewalls may struggle to handle scenarios such as network isolation between different tenants and access control between different services within the same tenant flexibly. Therefore, NexaSphere introduces a new component: security group. A security group is a distributed firewall focused on controlling east-west traffic and supports inbound and outbound traffic control at the virtual machine NIC level.

A security group consists of two logical sets:

- The set of security group rules: supports the addition, deletion, and modification of rules to control rule actions.
- The set of associated NICs: supports associating VM NICs with the security group so that its rules apply to those NICs.

Characteristics of security groups:

- Default policies: If no rules are added, except communication between group members, the security group denies all inbound traffic by default, and allows all outbound traffic.
- Rule flexibility: Rules support on-demand modification, including source IP, destination IP, destination port, protocol type, and priority.
- Dynamic priority adjustment: When adding a rule, you can insert it at a specific priority. When removing a rule, priorities automatically adjust to remain consecutive.
- Multiple security groups per NIC: A VM NIC can be associated with multiple security groups. You can dynamically adjust the priority of these groups. By default, rules in the security group associated first take effect first.

How It Works

Security groups have the following key points:

- Composition of a security group:

  - Source: Supports source data (for inbound direction) and destination data (for outbound direction).
  - Address Type: Supports IPv4 and IPv6.
  - Protocol Type and Port: Supports ICMP, TCP, UDP, and more.
  - Action: Deny or allow.
- Priority of security group rules:
  - Rule priority is a consecutive and unique (priority 0 indicates the highest priority by default). Lower priority numbers indicate higher precedence.
  - By default, newly added rules have the lowest priority. When inserting a rule at a specified priority, subsequent rule priorities adjust automatically to ensure uniqueness.
  - When traffic passes through a virtual machine NIC, matching starts from the rule with the highest priority. If a match succeeds, the rule action executes. Otherwise, matching continues with the next rule.
- Import and export of security group rules:
  - Existing rules support one-click export.
  - Exported rules supports re-importing into other security groups with rule validation provided during import.
- Associating multiple security groups to a NIC:
  - A VM NIC can be associated with multiple security groups. The associated groups support priority ordering. A lower number indicates a higher group priority.
  - When traffic passes through the virtual machine NIC, matching starts from the security group with the highest priority. If a rule within that group match succeeds, the rule action executes. Otherwise, matching proceeds to the next security group.
- NIC default policy:
  - By default, the NIC's default inbound policy is Allow, and the default outbound policy is Deny . These default policies support modification.
  - When traffic does not match any security group rule, the NIC's default policy executes.

Applications

1. Access control among different businesses.

   Take a typical web application as an example.

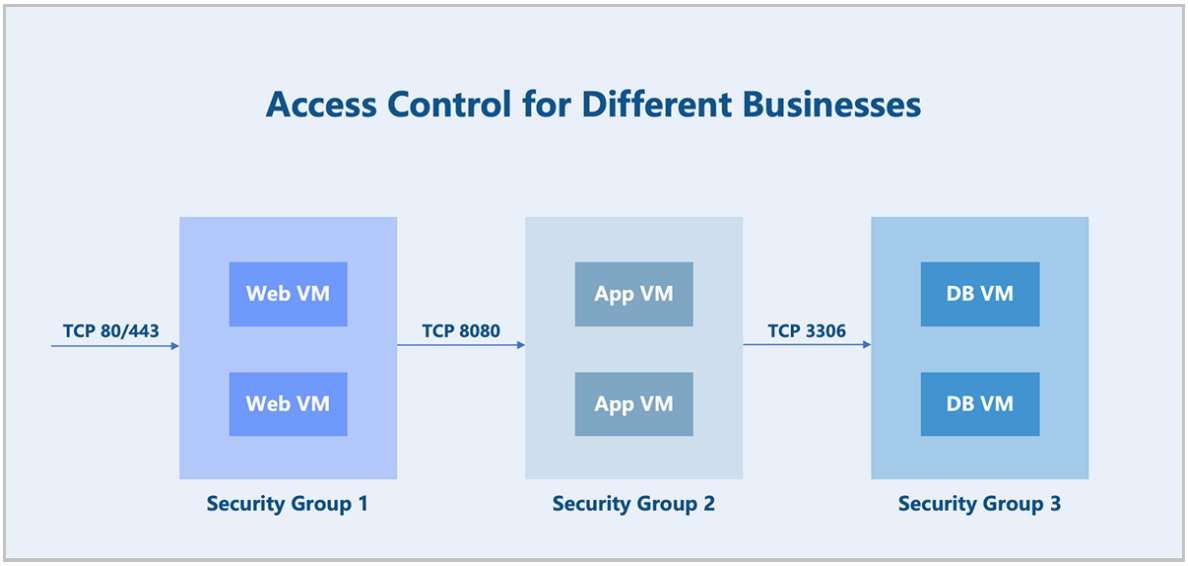
   - Allow any IP address to access TCP ports 80 and 443 on the Web virtual machines.

- Allow the Web virtual machines to access TCP port 8080 on the App virtual machines.
- Allow the App virtual machines to access TCP port 3306 on the DB virtual machines.

Create separate security groups for the three service tiers (Web, App, and DB) and configure the corresponding rules to control access between the tiers.

See *Figure 1-16: Access Control for Different Businesses* for an illustration.

Figure 1-16: Access Control for Different Businesses



2. Coexistence of multiple services. In addition to application traffic, the VMs in the preceding example must allow administrative access over SSH. You can implement this access in either of the following ways:

   - Option 1: Add a new rule to each of the three security groups.
   - Option 2: Create a new security group and associate it to the NICs of all the above business virtual machines.

   Because a VM NIC can be associated with multiple security groups, the second option provides more flexible control. Each service VM NIC can use one security group for application traffic and another for administrative access, with group priorities and default rules adjusted as needed.

3. Disabling communication within a security group.

   By default, VMs in the same security group can communicate without being restricted by security group rules. To isolate VMs within the same group, disable the **Allow Intra-Group Traffic** default rule. This rule cannot be deleted.

1.4.1.4 Virtual Resource Management

1.4.1.4.1 VM Management

1.4.1.4.1.1 VM Scheduling Policy

A VM scheduling policy defines how hosts are assigned to VMs to support service performanc e and availability. You can add VMs to a VM scheduling group and associate the group with a scheduling policy to control VM placement.

How It Works

NexaSphere supports the following scheduling relationships:

- VM-exclusive-from-each-other and VM-affinitive-to-each-other policies do not require a host scheduling group. VMs are assigned to hosts according to the policy and its execution mechanism.
- VMs-exclusive-from-hosts and VMs-affinitive-to-hosts policies require a corresponding host scheduling group. VMs are assigned to hosts according to the policy and its execution mechanism.

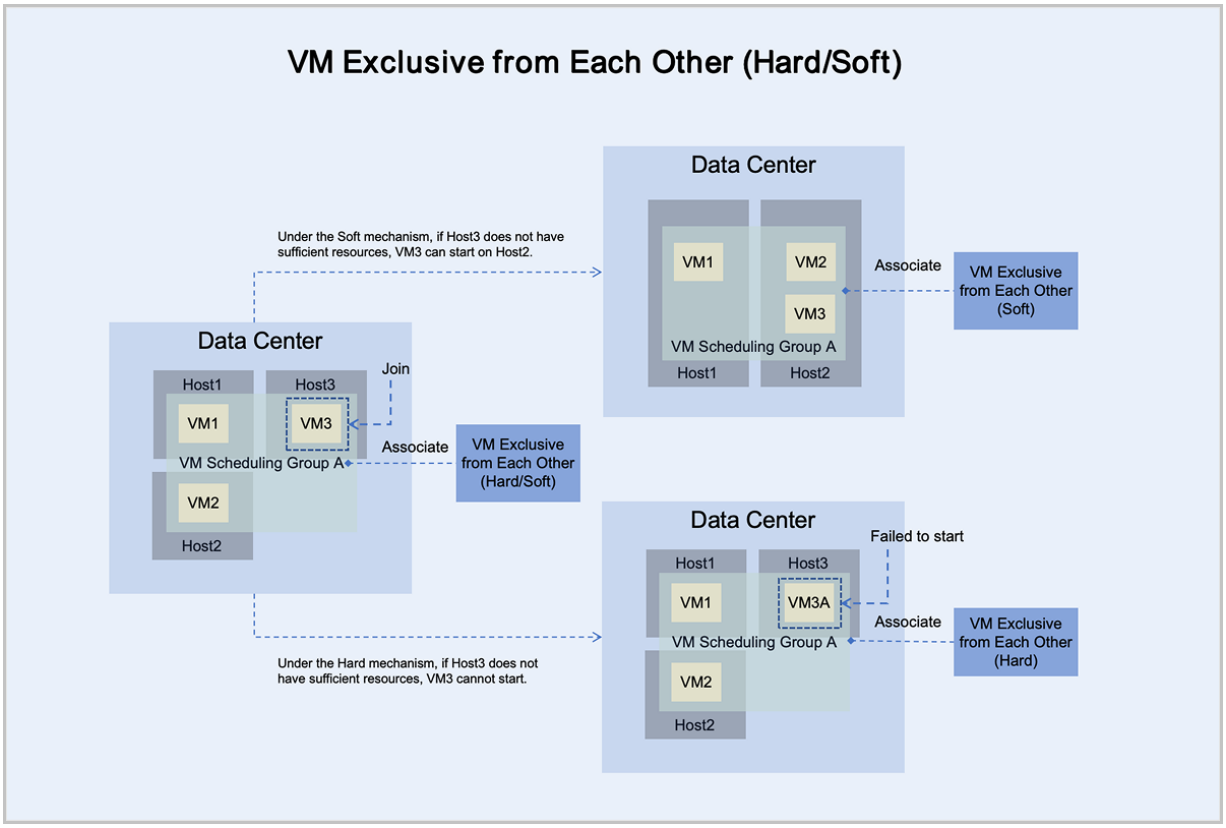
The following scenarios illustrate how the four scheduling policy types work.

Scenario 1: Assume that a data center contains Host1, Host2, and Host3. VM scheduling group A is associated with the **VM Exclusive from Each Other** policy. VM1 and VM2 belong to the group and run on Host1 and Host2, respectively. When VM3 joins the group, its behavior depends on the execution mechanism:

- With the Hard mechanism, VM3 must run on a different host from the other VMs in the group:
  - If Host3 has sufficient resources, VM3 starts and runs on Host3.
  - If Host3 has insufficient resources, VM3 cannot start.
- With the Soft mechanism, the scheduler prioritizes placing VM3 on a different host from the other VMs in the group:
  - If Host3 has sufficient resources, VM3 starts and runs on Host3.
  - If Host3 has insufficient resources, VM3 can start on another host with sufficient resources. In this scenario, VM3 starts and runs on Host2.

See *Figure 1-17: VM Exclusive from Each Other (Hard/Soft)* for an illustration.

Figure 1-17: VM Exclusive from Each Other (Hard/Soft)

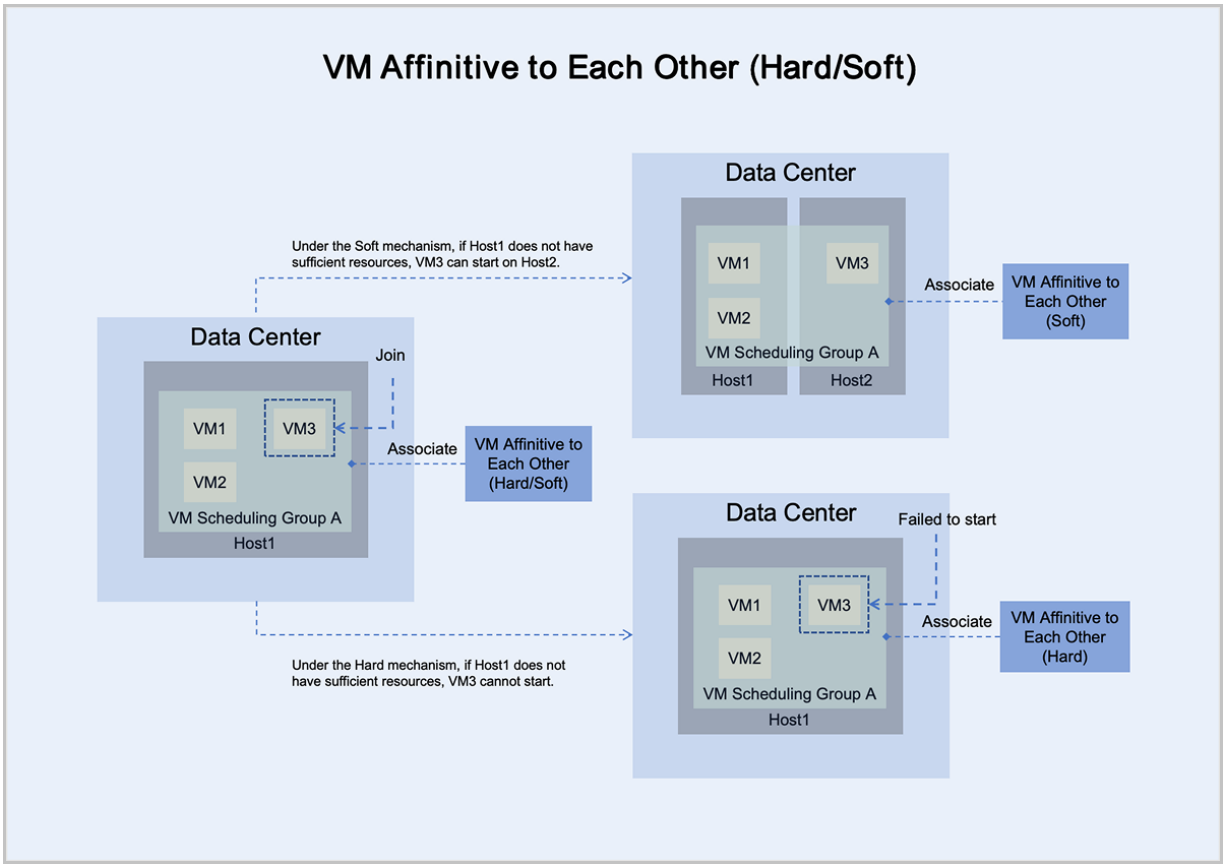


Scenario 2: Assume that a data center contains Host1 and Host2. VM scheduling group A is associated with the **VM Affinitive to Each Other** policy. VM1 and VM2 belong to the group and run on Host1. When VM3 joins the group, its behavior depends on the execution mechanism:

- With the Hard mechanism, VM3 must run on the same host as the other VMs in the group:
  - If Host1 has sufficient resources, VM3 starts and runs on Host1.
  - If Host1 has insufficient resources, VM3 cannot start.
- With the Soft mechanism, the scheduler prioritizes placing VM3 on the same host as the other VMs in the group:
  - If Host1 has sufficient resources, VM3 starts and runs on Host1.
  - If Host1 has insufficient resources, VM3 can start on another host with sufficient resources. In this scenario, VM3 starts and runs on Host2.

See *Figure 1-18: VM Affinitive to Each Other (Hard/Soft)* for an illustration.

Figure 1-18: VM Affinitive to Each Other (Hard/Soft)

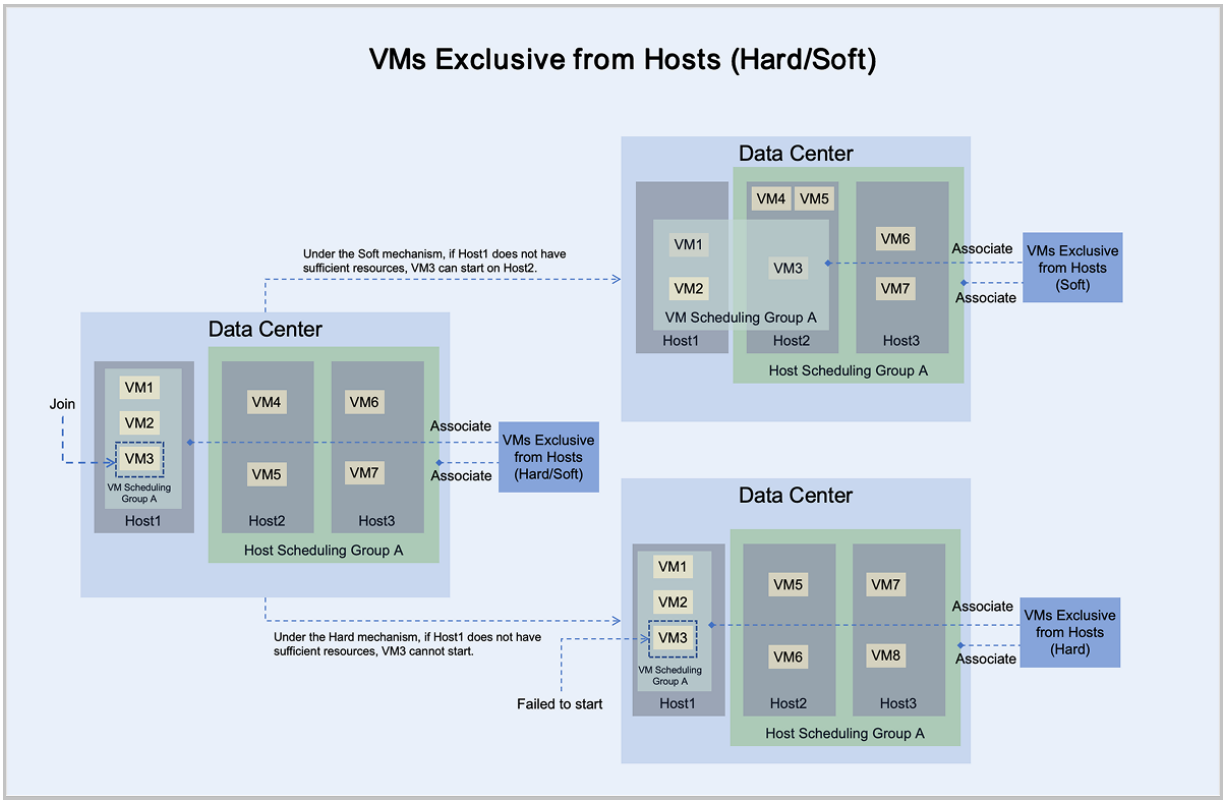


Scenario 3: Assume that a data center contains Host1, Host2, and Host3. VM scheduling group A is associated with the **VMs Exclusive from Hosts** policy. VM1 and VM2 belong to the group and run on Host1. Host scheduling group A is also associated with the **VMs Exclusive from Hosts** policy, and Host2 and Host3 belong to that group. When VM3 joins VM scheduling group A, its behavior depends on the execution mechanism:

- With the Hard mechanism, VM3 must not run on a host in host scheduling group A:
  - If Host1 has sufficient resources, VM3 starts and runs on Host1.
  - If Host1 has insufficient resources, VM3 cannot start.
- With the Soft mechanism, the scheduler prioritizes placing VM3 on a host outside host scheduling group A:
  - If Host1 has sufficient resources, VM3 starts and runs on Host1.
  - If Host1 has insufficient resources, VM3 can start on another host with sufficient resources. In this scenario, VM3 starts and runs on Host2.

See *Figure 1-19: VMs Exclusive from Hosts (Hard/Soft)* for an illustration.

Figure 1-19: VMs Exclusive from Hosts (Hard/Soft)

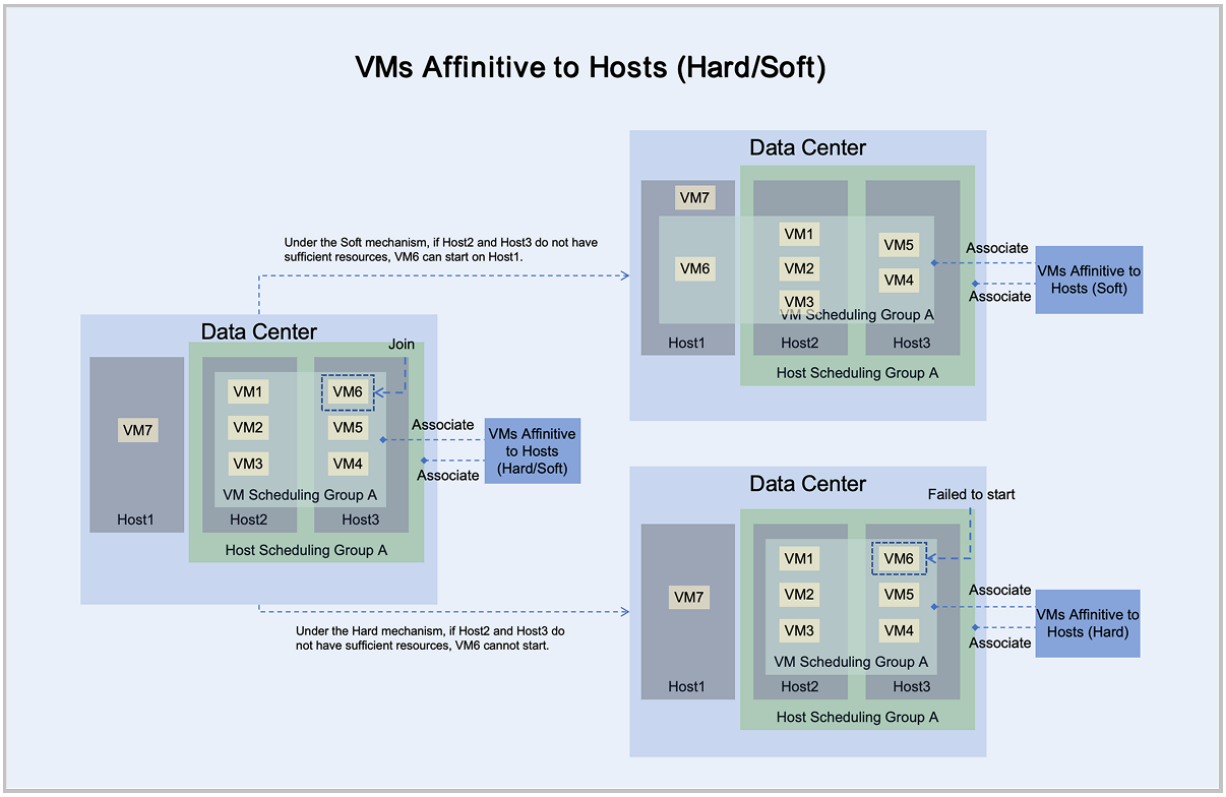


Scenario 4: Assume that a data center contains Host1, Host2, and Host3. VM scheduling group A is associated with the **VMs Affinitive to Hosts** policy. VM1 through VM5 belong to the group and run on Host2 and Host3. Host scheduling group A is also associated with the **VMs Affinitive to Hosts** policy, and Host2 and Host3 belong to that group. When VM6 joins VM scheduling group A, its behavior depends on the execution mechanism:

- With the Hard mechanism, VM6 must run on a host in host scheduling group A:
  - If Host2 or Host3 has sufficient resources, VM6 starts and runs on that host.
  - If both Host2 and Host3 have insufficient resources, VM6 cannot start.
- With the Soft mechanism, the scheduler prioritizes placing VM6 on Host2 or Host3:
  - If Host2 or Host3 has sufficient resources, VM6 starts and runs on that host.
  - If both Host2 and Host3 have insufficient resources, VM6 can start on another host with sufficient resources. In this scenario, VM6 starts and runs on Host1.

See *Figure 1-20: VMs Affinitive to Hosts (Hard/Soft)* for an illustration.

Figure 1-20: VMs Affinitive to Hosts (Hard/Soft)



1.4.1.4.1.2 VM Cloning

NexaSphere provides two cloning methods to meet different service needs: full clone and instant full clone.

Full Clone

A full clone does not share data with the source VM. The source VM is first converted into an image file that contains the files and configuration required by the VM and is stored in image storage. The image is then pushed to data storage as an image cache from which the cloned VM is created. The cloned VM can use data storage different from that of the source VM.

A fully cloned VM is isolated from and independent of its source, so the source VM does not affect its performance. Full cloning typically consumes more storage. However, NexaSphere optimizes batch full-clone operations by storing only one image cache on each data storage. This reduces storage consumption and accelerates the startup of cloned VMs.

Instant Full Clone

Instant full cloning first uses linked-clone technology to create a VM that temporarily depends on the source VM, enabling fast provisioning. A background task then merges snapshots to separate

the cloned VM's data completely from the source. The cloned VM can start quickly and becomes fully independent after the merge completes, with no subsequent performance impact from the source VM.

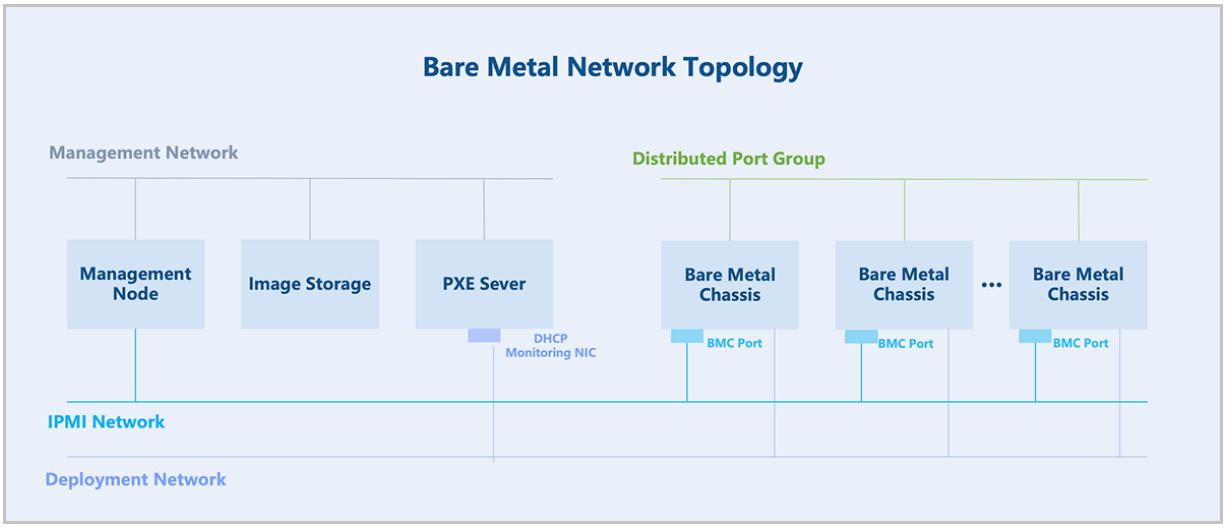
1.4.1.4.2 Bare Metal Management

The network traffic model for bare metal management includes the management network, distributed port group, IPMI network, and deployment network.

- Management Network: Used to manage hardware resources associated with the platform.
- Distributed Port Group: Serves as the service network for bare metal instances to provide application services.
- IPMI Network: Used by the management node to perform remote operations on bare metal chassis and bare metal instances, such as powering on/off, rebooting, and retrieving hardware information.
- Deployment Network: Used by the PXE sever to assign IP addresses via the DHCP service and to transfer images via the TFTP service.

See *Figure 1-21: Bare Metal Network Topology* for an illustration.

Figure 1-21: Bare Metal Network Topology



The core of the bare metal management: hardware information retrieval and unattended deployment of bare metal chassis.

Bare metal management workflow:

1. The management node directs the bare metal chassis to perform a PXE boot via the PXE server.

2. The PXE server provides DHCP, TFTP, and image storage services. After the bare metal device boots through PXE, it obtains an IP address through DHCP, downloads **pxelinux.0** and boot files from the TFTP server, loads the kernel into memory, and boots into a LiveCD system.
3. In the LiveCD system, a detection script runs and reports the hardware information of the bare metal chassis back to the management node.
4. Based on the returned hardware information, a preconfigured template is applied to the bare metal chassis. This template includes partition information, NIC bonding, IP address, and more.
5. Select an OS ISO for installation to deploy the bare metal instance.
6. The bare metal chassis is rebooted for a PXE boot. The PXE server pre-downloads the target OS ISO, and the bare metal chassis performs an unattended deployment according to the preconfigured template. After deployment, the chassis automatically configures the NIC and other settings based on the preconfigured template, completing the bare metal instance configuration.
7. To simplify operations management for bare metal hosts, the PXE server can deploy a monitoring service that collects CPU, memory, disk capacity, disk I/O, NIC, and other metrics from the hosts in real time.

1.4.2 Data Protection

1.4.2.1 Snapshot Management

NexaSphere supports both Redirect-On-Write (ROW) and Copy-On-Write (COW) snapshot mechanisms.

- For centralized storage: Local storage, NFS, and SAN storage use QCOW2 external snapshot, which is a type of ROW snapshot mechanism.
- For distributed storage: Enterprise Ceph storage uses ROW snapshots. nSAN uses COW snapshots.

Centralized Storage Snapshot Mechanism

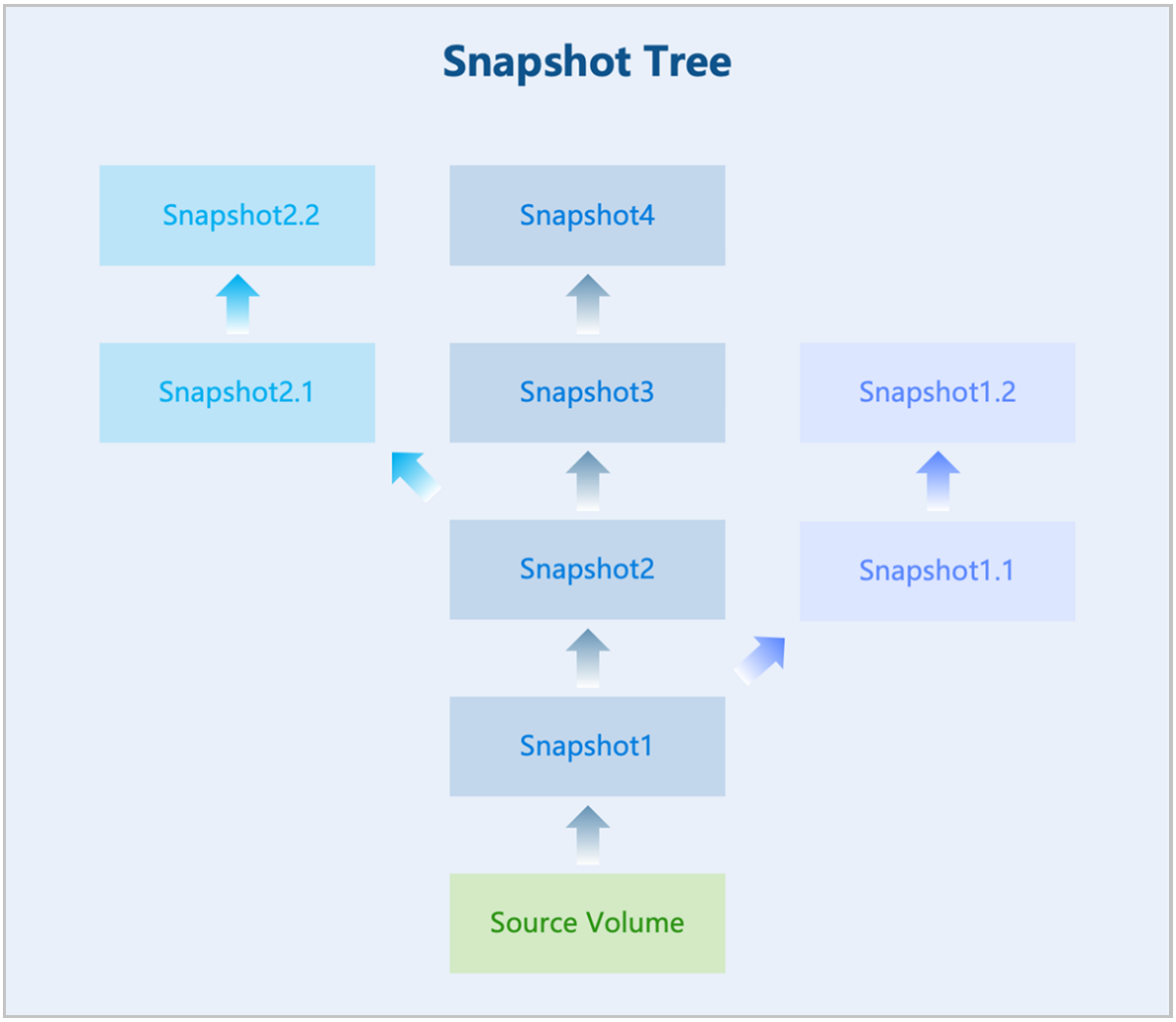
This section introduces QCOW2 external snapshots.

1. **Snapshot Chain and Snapshot Tree**

   A disk typically has one snapshot chain and can have one snapshot tree. Each branch of the snapshot tree is a snapshot chain.

   See *Figure 1-22: Snapshot Tree* for an illustration.

Figure 1-22: Snapshot Tree



A snapshot tree includes the following information:

- Snapshot Chain: A relational chain composed of a set of snapshots for a disk. Each branch of the snapshot tree is a snapshot chain.
- Snapshot Node: An individual node in a snapshot chain, representing a single snapshot of the disk.
- Snapshot Size: The storage space consumed by a snapshot. You can view the total size of all snapshots in the snapshot tree and the size of an individual snapshot node.

Note:

- For non-distributed storage, the system allows a maximum of 128 nodes per snapshot chain by default. For distributed storage, each disk supports up to 32 snapshots, including manual and automatic snapshots.

- When a snapshot chain reaches its maximum length:
  - If you continue to create automatic snapshots, the system automatically deletes the earliest automatic snapshot.
  - If you continue to create manual snapshots, you must manually delete unneeded snapshots.
- In production, keep the number of snapshots for each VM at five or fewer whenever possible. Excessive snapshots can affect VM performance, data security, and data storage capacity.

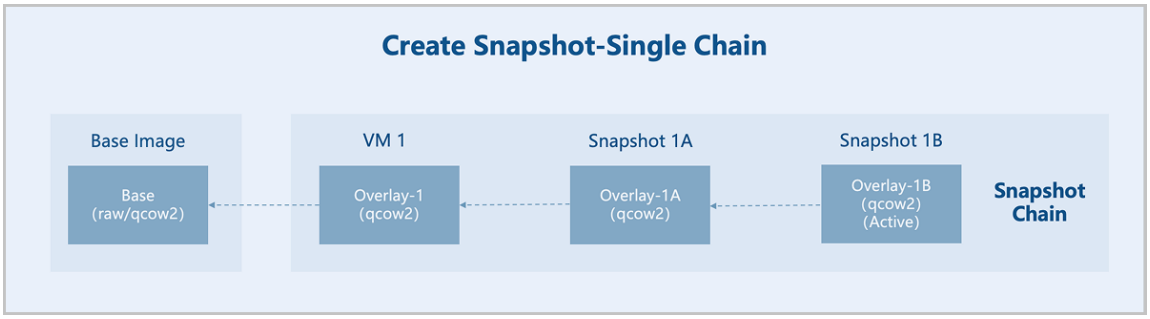
2. Create Snapshots

When an external snapshot is created, a new empty QCOW2 file is generated. This new file has its backing file pointed to the original QCOW2 file. The original QCOW2 file is set to read-only, effectively turning it into a snapshot itself. Subsequent data writes are directed only to the new QCOW2 file. The creation of an external snapshot involves creating a new blank qcow2 file.

- Create a single snapshot chain based on a backing file.

  See *Figure 1-23: Single Snapshot Chain* for an illustration.

  Figure 1-23: Single Snapshot Chain



  Assume there is an original base image (Base). A virtual machine 1 is created using this base image as a template. Then, snapshot 1A and 1B are created sequentially for virtual machine 1.

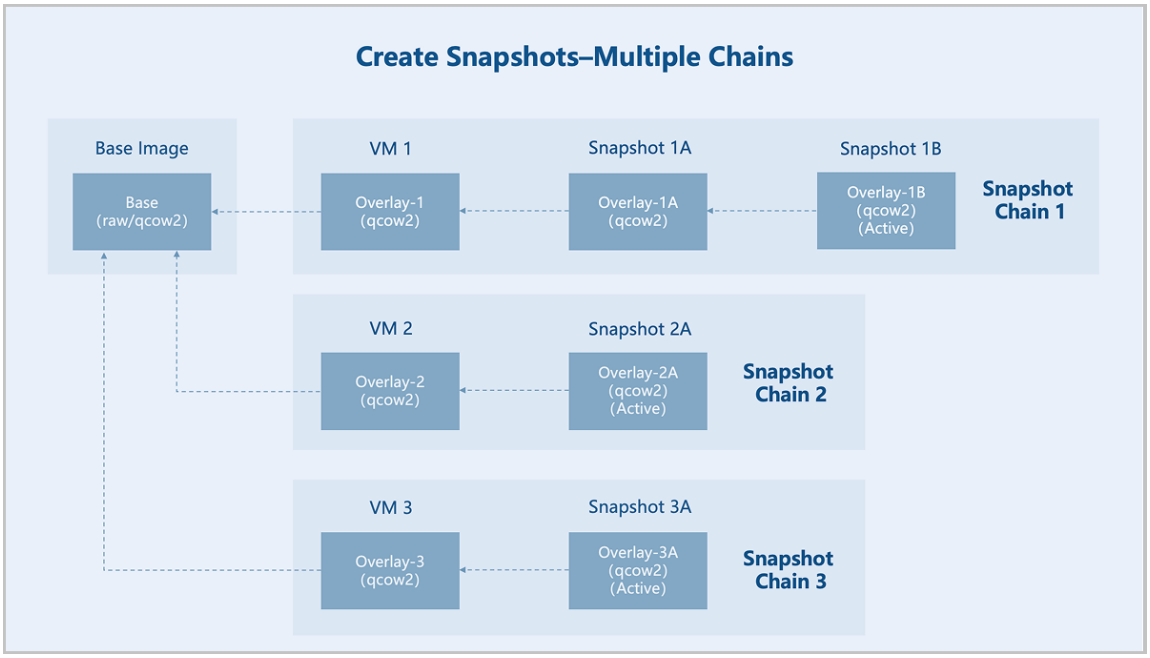
  - Base Image: A pre-made disk image file containing a complete operating system and bootloader, serving as the Base (read-only).
  - VM 1: A new empty file, Overlay-1, is created. Its backing file points to Base. Base remains read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1.

- Snapshot 1A: A new empty file, Overlay-1A, is created. Its backing file points to Overlay -1. Overlay-1 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1A.
- Snapshot 1B: A new empty file, Overlay-1B, is created. Its backing file points to Overlay -1A. Overlay-1A is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1B. VM 1 uses the disk file corresponding to the last snapshot, Snapshot 1B, which is the active snapshot.

- Create multiple snapshot chains based on backing files.

  See *Figure 1-24: Multiple Snapshot Chains* for an illustration.

Figure 1-24: Multiple Snapshot Chains



Assume that VM 1, VM 2, and VM 3 are created from an original base image (Base). Snapshot 1A and Snapshot 1B are then created for VM 1, Snapshot 2A for VM 2, and Snapshot 3A for VM 3.

- Base Image: A pre-made disk image file containing a complete operating system and bootloader, serving as the Base (read-only).
- Snapshot Chain 1:
  - Virtual Machine 1: A new empty file, Overlay-1, is created. Its backing file points to Base. Base remains read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1.

- Snapshot 1A: A new empty file, Overlay-1A, is created. Its backing file points to Overlay-1. Overlay-1 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1A.
- Snapshot 1B: A new empty file, Overlay-1B, is created. Its backing file points to Overlay-1A. Overlay-1A is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-1B. VM 1 uses the disk file corresponding to the last snapshot 1B in snapshot chain 1, and snapshot 1B is Active.

- Snapshot Chain 2:
  - Virtual Machine 2: A new empty file, Overlay-2, is created. Its backing file points to Base. Base remains read-only. Subsequent data is written only to Overlay-2.
  - Snapshot 2A: A new empty file, Overlay-2A, is created. Its backing file points to Overlay-2. Overlay-2 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-2A. VM 2 uses the disk file corresponding to the last snapshot 2A in snapshot chain 2, and snapshot 2A is Active.
- Snapshot Chain 3:
  - Virtual Machine 3: A new empty file, Overlay-3, is created. Its backing file points to Base. Base remains read-only. Subsequent data is written only to Overlay-3.
  - Snapshot 3A: A new empty file, Overlay-3A, is created. Its backing file points to Overlay-3. Overlay-3 is set to read-only, thus becoming a snapshot. Subsequent data is written only to Overlay-3A. VM 3 uses the disk file corresponding to the last snapshot 3A in snapshot chain 3, and snapshot 3A is Active.

3. Merge Snapshots

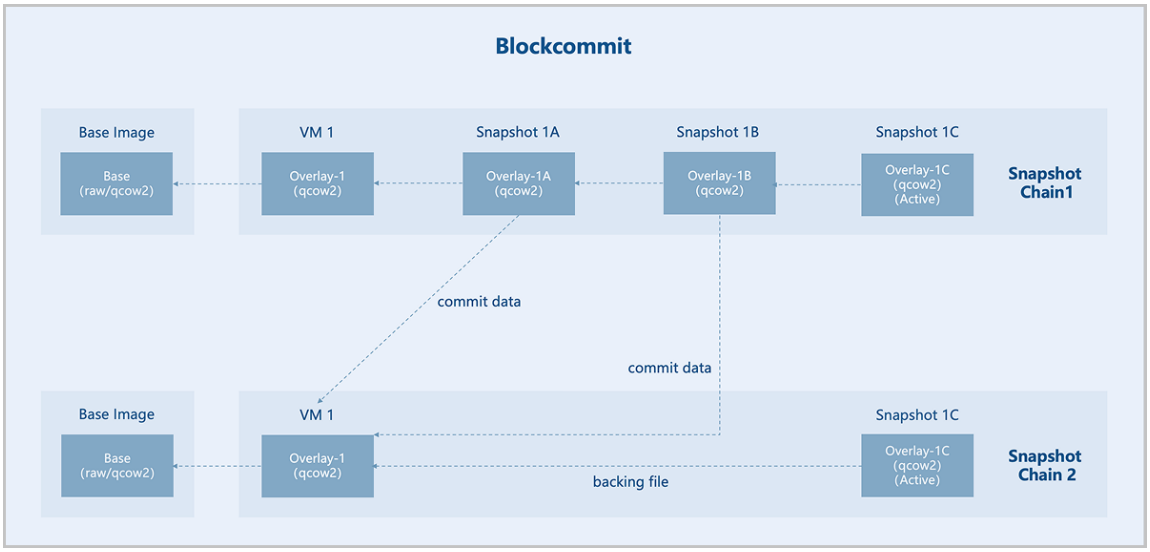
External snapshots are interdependent, where each overlay file depends on its backing file. Each snapshot preserves its corresponding data, preventing the direct deletion of an individual snapshot to shorten the chain length. The chain length of external snapshots can be reduced through two methods: downward merging (Blockcommit) or upward merging (Blockpull).

- Blockcommit

  Within the same snapshot chain, you can merge overlays into backing files.

  See *Figure 1-25: Downward Merge (Blockcommit)* for an illustration.

Figure 1-25: Downward Merge (Blockcommit)



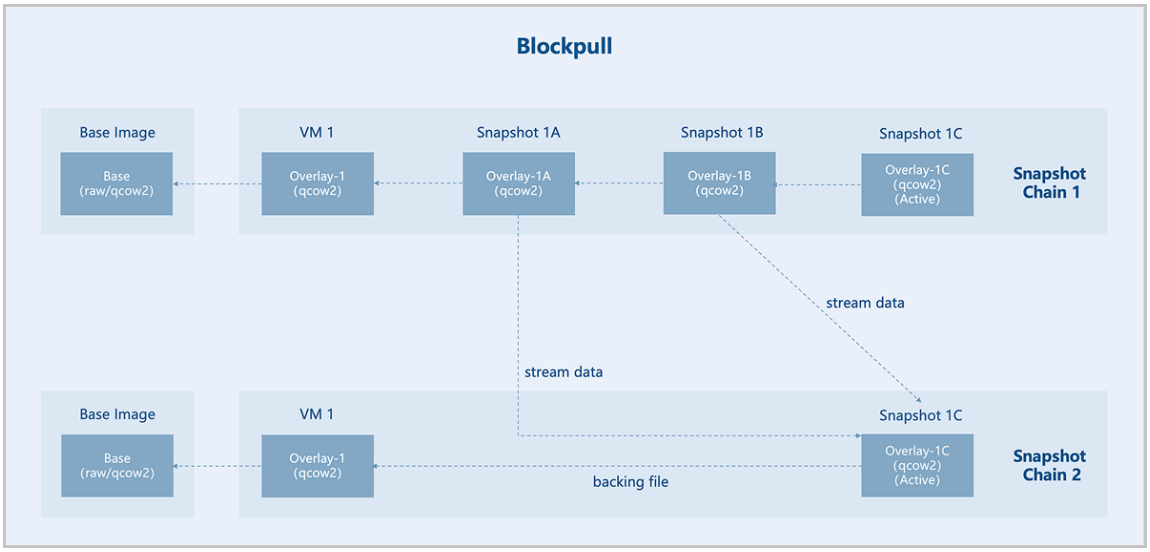
Assume there is an original base image (Base). Virtual machine 1 is created based on Base , and three interdependent external snapshots are created for virtual machine 1: Snapshot 1A, Snapshot 1B, and Snapshot 1C. Now, Snapshot 1A and Snapshot 1B are merged downward into virtual machine 1. As a result, the backing file of Snapshot 1C (Active) points directly to virtual machine 1, so the snapshot chain is shortened. Snapshots 1A and 1B are no longer useful and can be deleted.

- Blockpull

  Within the same snapshot chain, you can merge backing files into overlays.

  See *Figure 1-26: Upward Merge (Blockpull)* for an illustration.

Figure 1-26: Upward Merge (Blockpull)



Assume there is an original base image (Base). Virtual machine 1 is created based on Base , and three interdependent external snapshots are created for virtual machine 1: Snapshot 1A, Snapshot 1B, and Snapshot 1C. Now, Snapshot 1A and Snapshot 1B are merged upward into Snapshot 1C (Active). As a result, the backing file of Snapshot 1C (Active) points directly to virtual machine 1, so the snapshot chain is shortened. Snapshots 1A and 1B are no longer useful and can be deleted.

1.4.2.2 Backup and Recovery

1.4.2.2.1 Data Backup

The backup service performs data backup at the QEMU block-device layer and supports VMs on all data storage types. A full backup contains the complete data set, while an incremental backup contains only data changed since the preceding backup. Both backup types copy only data that is actually in use.

By default, after the initial full backup, the backup following every 63 incremental backups is another full backup. Restoring a VM requires the selected recovery point, all preceding incrementa l backups, and the full backup on which they depend. The maximum backup-chain length is therefore 64. The system also selects an appropriate backup method according to the current backup state to protect backup-data integrity.

The data backup process consists of data replication, data transfer, and data retention.

1.4.2.2.1.1 Data Replication

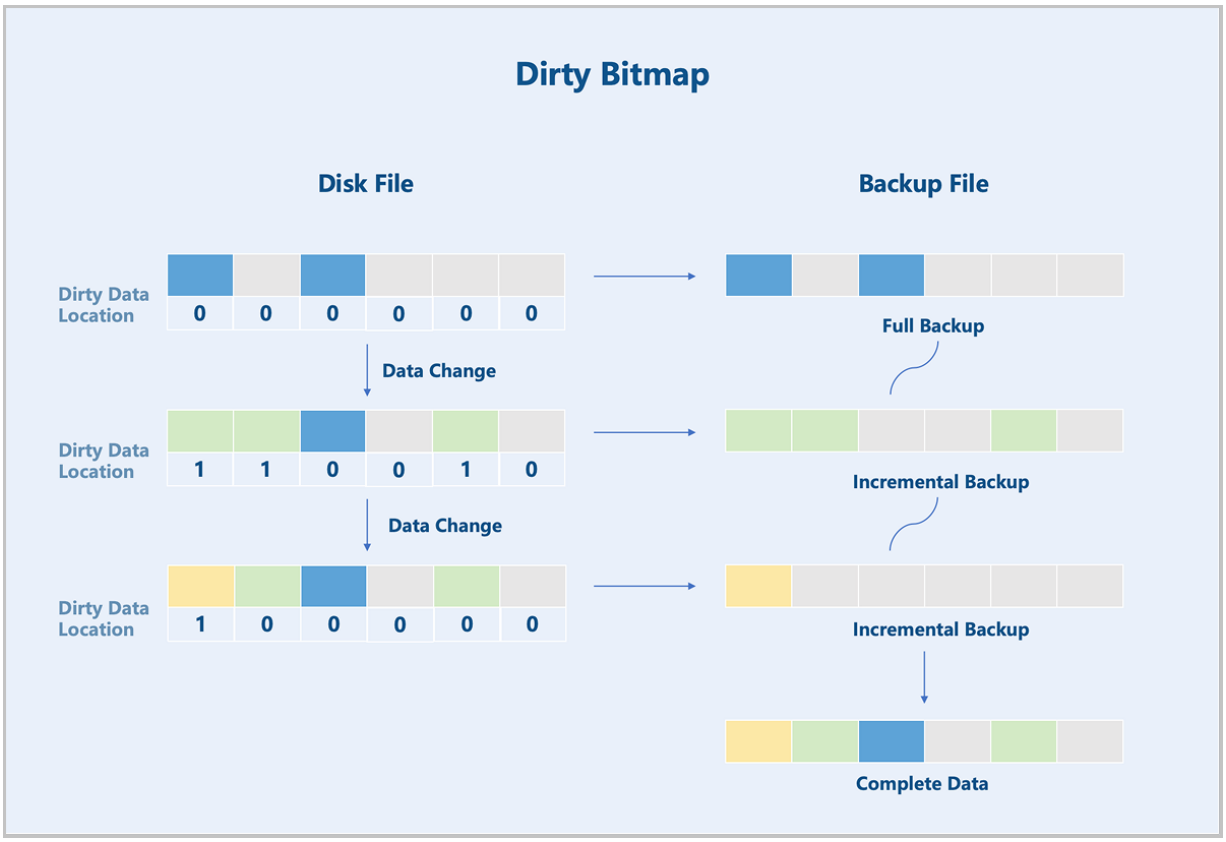
The backup service uses the dirty-data tracking capability (Dirty Bitmap) at the QEMU block-device layer to export backup data.

Locations in the virtual machine's disk where data changes occur are called dirty data locations. The Dirty Bitmap records all such locations in the virtual disk file that have been modified since the last backup. Based on these records, all data modified since the last backup (incremental backup data) can be exported.

The full backup and subsequent incremental backups form a complete backup chain. Because the Dirty Bitmap is held in the memory of the QEMU process, its tracking data is lost when the VM restarts. The first backup after a VM restart is therefore automatically performed as a full backup.

See *Figure 1-27: Dirty Bitmap* for an illustration.

Figure 1-27: Dirty Bitmap



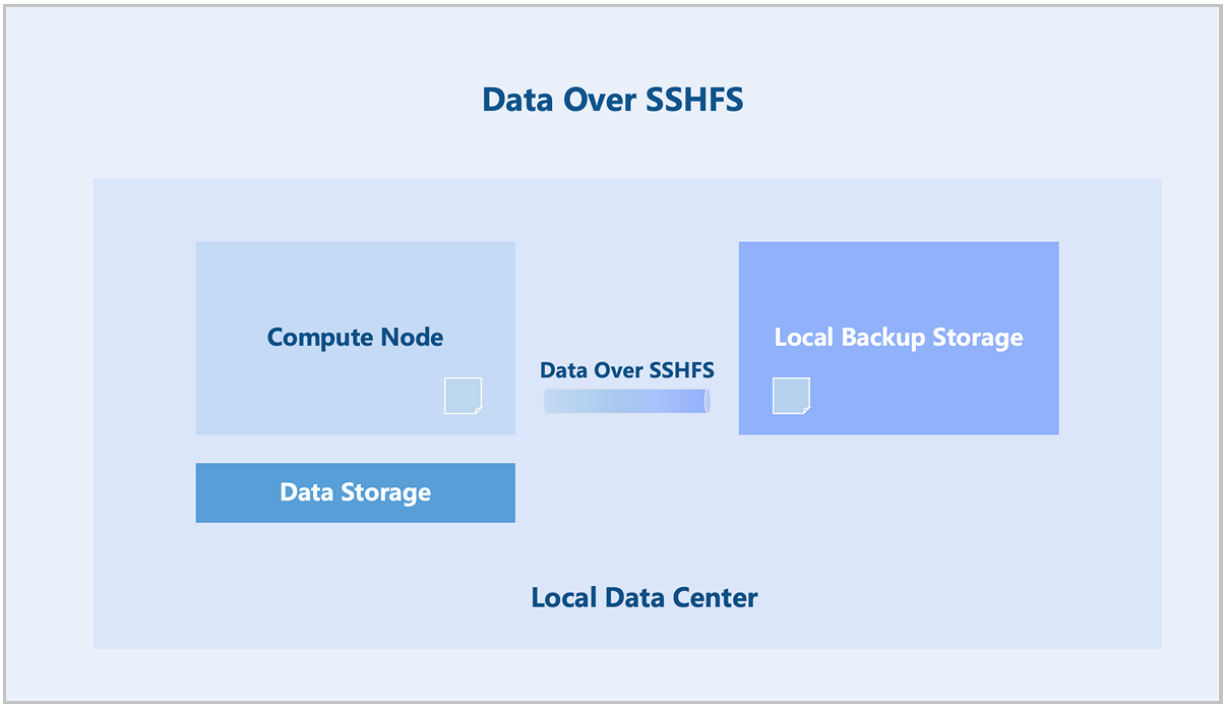
1.4.2.2.1.2 Data Transfer

Two implementations are available for different virtualization component versions. They differ mainly in how backup data is transferred.

Scheme 1: Data over SSHFS. SSHFS mounts the backup directory of remote backup storage on the compute node, which then transfers backup data into backup storage. SSHFS is a FUSE-over-SSH implementation. The SSH session encrypts the data channel, and each backup plan uses a dedicated SSHFS connection.

See *Figure 1-28: Data Over SSHFS* for an illustration.

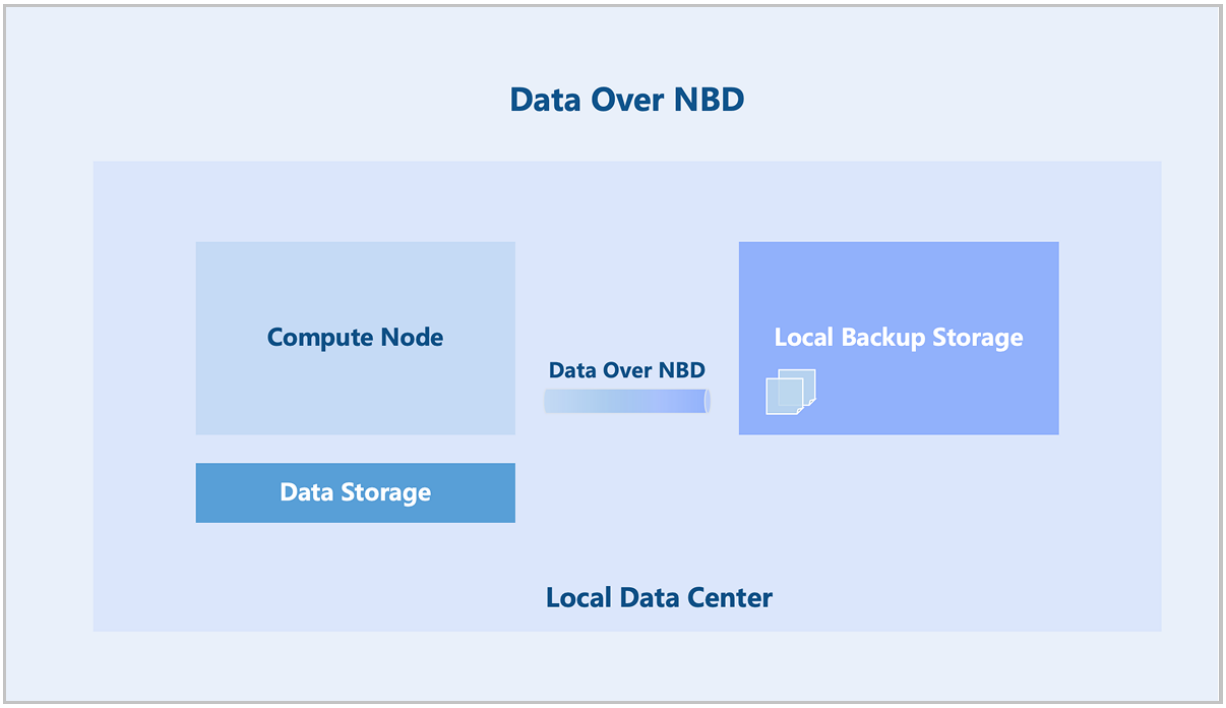
Figure 1-28: Data Over SSHFS



Scheme 2: Data over NBD. The NBD module on backup storage exports a backup disk. A QEMU block job on the compute node then writes backup data directly to that disk.

See *Figure 1-29: Data Over NBD* for an illustration.

Figure 1-29: Data Over NBD



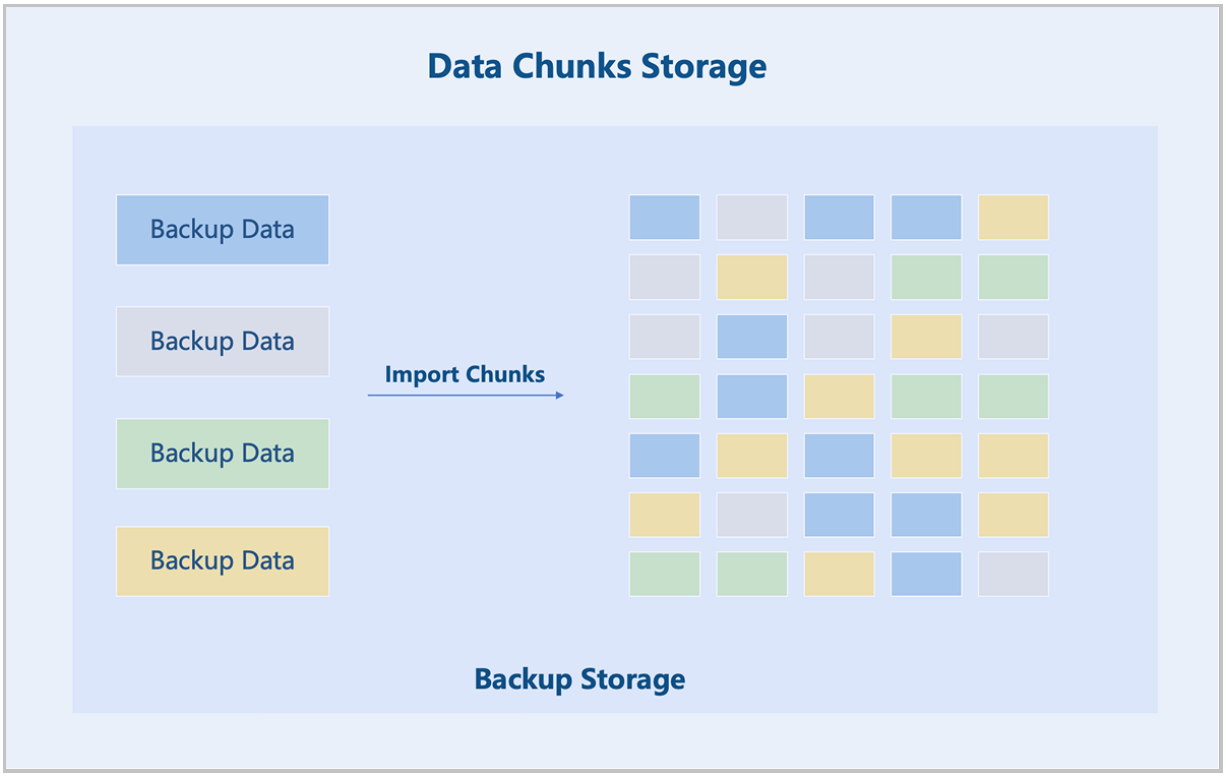
1.4.2.2.1.3 Data Retention

The backup storage supports various storage media types, including SAN, NAS, disk arrays, and tape libraries.

Backup storage retains data in deduplicated chunks. Backup data is split into 64 MB chunks, and a hash and index are generated for each chunk. Chunks with identical hash values are stored only once.

See *Figure 1-30: Chunked Data Retention* for an illustration.

Figure 1-30: Chunked Data Retention

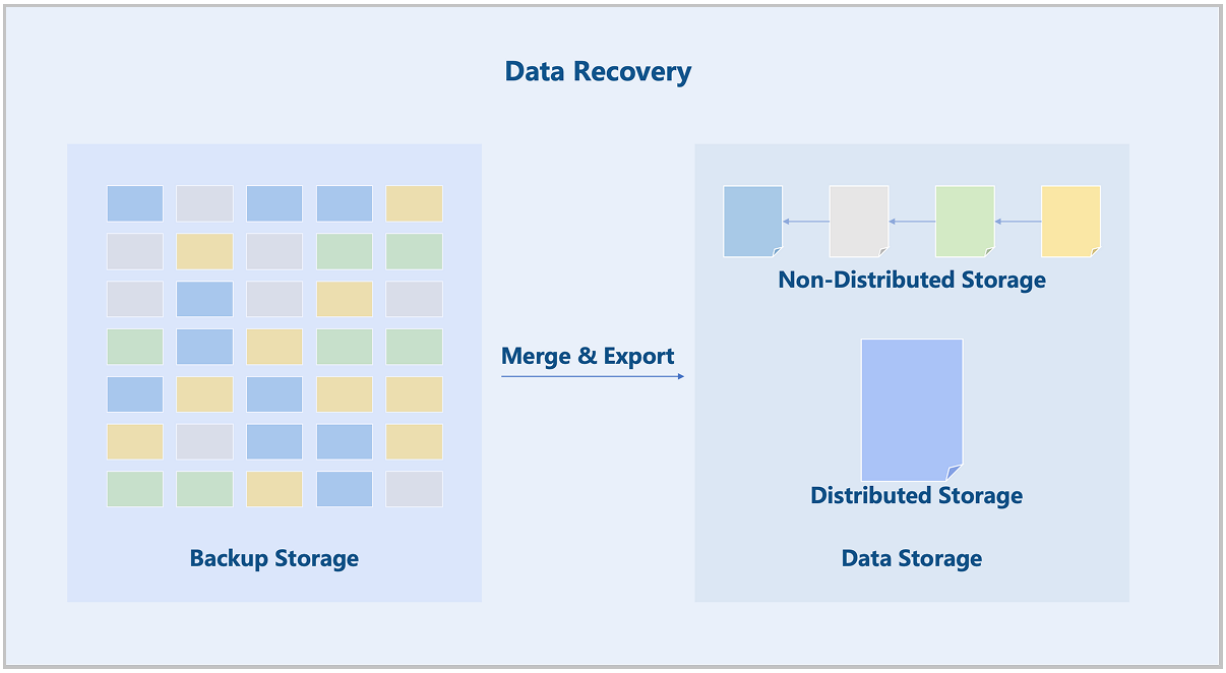


1.4.2.2.2 Data Recovery

When a VM is restored from a local backup, the chunked data in backup storage is merged and imported into data storage. On non-distributed storage, the restored data is retained as a disk chain. On distributed storage, the disk chain is merged into a single disk file. For a new recovery, the restored disk data serves as an image cache from which a new VM is created. For an overwrite recovery, the restored disk path replaces the current path in the VM database record , after which the old VM files are deleted.

See *Figure 1-31: Data Recovery* for an illustration.

Figure 1-31: Data Recovery



1.4.3 Operations Management

1.4.3.1 Management Node Monitoring

In a multi-management-node high-availability environment, you can view the health of each management node.

Management node monitoring displays the management node IP address, node status, VIP, and management service status for each node. The monitored services include:

- Arbiter Gateway Reachable: Indicates whether the arbitration IP address used by the active and standby management nodes is reachable. If it is unreachable, management node high availability might fail.
- Peer MN Reachable: Indicates whether the peer management node is reachable. If it is unreachable, the management nodes cannot communicate with each other.
- VIP Reachable: Indicates whether the VIP is reachable. If it is unreachable, the management UI cannot be accessed through the VIP.
- Database Status: Indicates whether the database is healthy and synchronized across management nodes. An unhealthy or unsynchronized database can result in data loss and requires prompt recovery.

1.4.3.2 Monitoring and Alarms

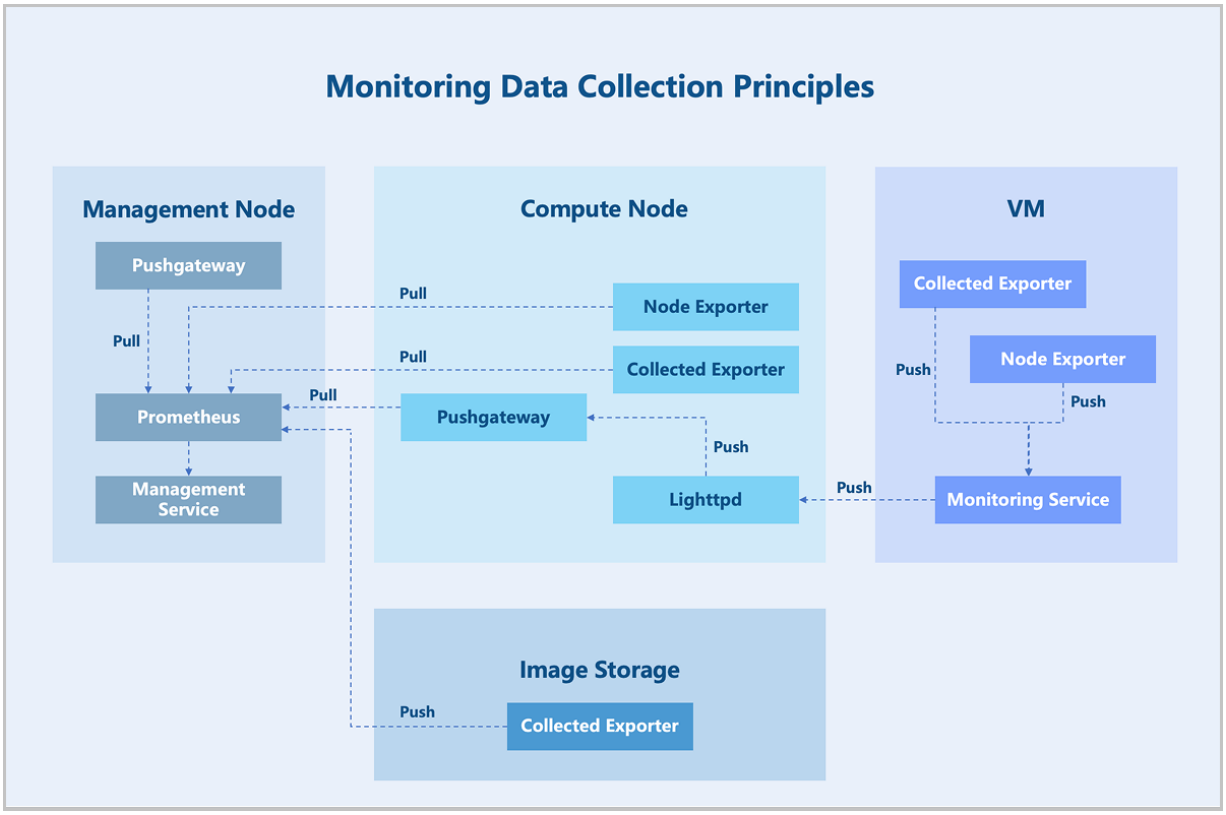
Monitoring and alarms track time-series data, such as resource utilization and capacity metrics , as well as predefined system events. The notification service (SNS) sends alarm messages to configured endpoints. Supported alarm types include resource alarms and event alarms. Endpoints include the system message center, email, DingTalk, WeCom, Lark, Webhook, SMS, Microsoft Teams, and SNMP Trap. Some resource alarms require an agent.

Prometheus provides time-series monitoring. It aggregates metrics from different sources and collects them through a unified service.

In this architecture, the Prometheus server collects and stores data and provides query access instead of directly generating metrics for each target. Exporters periodically collect samples such as host CPU utilization. NexaSphere uses both pull and push modes according to the monitoring target. For hosts and metrics collected outside VMs, Prometheus periodically pulls the data gathered by exporters. Network or security restrictions can prevent Prometheus from directly reaching services inside VMs . In that case, a Pushgateway relays the data: exporters collect the metrics and periodically push them to the Pushgateway, and Prometheus then pulls the data from the Pushgateway for unified storage and querying.

See *Figure 1-32: Monitoring Data Collection Principles* for an illustration.

Figure 1-32: Monitoring Data Collection Principles



1.5 Reliability

1.5.1 Compute High Availability

NexaSphere uses KVM-based hardware virtualization to pool a group of physical servers and manage them as clusters. NexaSphere continuously monitors the operating status of every host and VM in a cluster. When a host fails, the NexaSphere management node confirms that the host is unavailable and restarts the affected VMs on another host in the cluster to maintain service continuity. NexaSphere compute high availability requires no dedicated standby hardware or third-party failover software, reducing downtime and service interruption without the cost and complexity of operating-system-specific or application-specific failover solutions.

1.5.2 Network High Availability

Virtual Network High Availability

NIC bonding combines two or more physical NICs into one logical interface. NexaSphere supports two bonding modes:

- Link aggregation (mode 4): Aggregates 1 to 8 physical NICs with the same speed and duplex settings. Traffic is distributed across the NICs to balance utilization.

  Link aggregation selects an egress NIC by hashing packet attributes. Three hash policies are available:

  - layer2+3: Uses the source MAC address, destination MAC address, and IP addresses to select the egress NIC.
  - layer3+4: Uses IP addresses and ports to select the egress NIC. This policy supports the TCP/IP stack.
  - layer2: Uses the source and destination MAC addresses to select the egress NIC.
- Active-backup (mode 1): Aggregates 1 to 8 physical NICs. One NIC is active and the others remain on standby. The active NIC carries all traffic. If it fails, a standby NIC takes over automatically to prevent service interruption.

1.5.3 Management Node High Availability

The management node controls, monitors, schedules, allocates, and reclaims resources in NexaSphere. If the management node becomes unavailable, management services stop, affecting NexaSphere operations, monitoring and alarms, user access, and automated tasks. High availability for NexaSphere management services therefore protects day-to-day platform operations.

Dual-Management-Node High Availability

NexaSphere provides a dual-management-node high-availability solution with the following characteristics:

1. If either management node loses power, the system continues operating and the UI remains available for task execution.
2. After the failed node recovers, its databases, including MySQL and the monitoring database, synchronize automatically.
3. The system can operate on a single management node for an extended period.
4. Deployment and operation of dual management nodes remain comparable to those of a single management node.

An HA process runs on each management node and initializes, configures, monitors, and maintains the management-node environment.

- The HA process provides a command-line interface for configuring the system as a high-availability environment.
- It monitors critical services, including the management node process, UI process, and MySQL . If a service fails, Keepalived triggers VIP failover and the HA process attempts to recover the failed service.
- It provides a watchdog for Keepalived to ensure that Keepalived remains running.
- It provides commands for displaying cluster health information.
- A gateway serves as an arbitration endpoint to prevent split-brain between the two management nodes.

Management Node Monitoring

Management node monitoring displays the management node IP address, node status, VIP, and management service status for each node. For details, see *Management Node Monitoring*.

Management Node Monitoring Data Collection

For the monitoring-data collection mechanism, see *Monitoring and Alarms*.

1.6 Security

1.6.1 Compute Security

1.6.1.1 HTTPS-encrypted UI Login

The system uses HTTPS by default for accessing the UI management interface, providing enhanced security.

- **HTTPS Protocol**: NexaSphere automatically redirects to the HTTPS address on port 443 by default. You can access the UI management interface by entering the management node IP address in your browser, for example, *https://management_node_ip*.
- **HTTP Protocol**: HTTP protocol is disabled by default, but you can configure it manually if needed.
- The system supports **PKCS12** format certificates by default. Currently, only PKCS12 and JKS certificates are supported. Convert certificates in other formats before importing them.

1.6.1.2 VM Console

The VM console provides users with a streamlined entry point for monitoring and managing virtual machines. You must have appropriate permissions to access the VM console. Two authentication methods are supported for console login: SSH key authentication and username/password.

- SSH Key Authentication
  - You can use SSH key authentication to log in to Linux virtual machines.
  - An SSH key pair consists of a **public key**, which can be shared, and a **private key**, which must be kept secure.
  - After a public key is attached to a virtual machine, you can use the corresponding private key to SSH into the virtual machine from another virtual machine without requiring a password.
  - To attach a public key during the virtual machine creation, ensure that the VM image has cloud-init pre-installed. The recommended cloud-init versions are 0.7.9, 17.1, 19.4, or later.
  - To attach a public key after the virtual machine creation, ensure that the virtual machine is running and has QEMU Guest Agent (QGA) installed and running. You can install QGA by installing the VMTools. If you install QGA by using other methods, install version 2.5 or later.
- Username/Password
  - You can log into virtual machines using a username and password.
  - The fixed username for Linux virtual machines is **root**, and the fixed username for Windows virtual machines is **administrator**.
  - After a password is injected into a VM, you can use the username and password to log in to the VM over SSH from another VM.
  - Ensure that the VM image has cloud-init pre-installed. The recommended cloud-init versions are 0.7.9, 17.1, 19.4, or later.

1.6.1.3 High Availability

VM HA

Virtual machines support the high availability (HA) mode. This policy can trigger automatic VM restart when a VM is stopped due to routine maintenance (planned) or unexpected failures (unplanned), thereby improving VM availability.

NeverStop VM HA Mechanism:

- The system uses polling and trigger-based mechanisms to monitor the virtual machine status. If the virtual machine is confirmed to be stopped, a VM configured with HA will be automatically restarted.
- The system uses polling and trigger-based mechanisms to monitor the virtual machine status. If the VM status cannot be definitively determined, the following detection process is initiated:

1. Based on the existing network configuration, select the most accurate method to probe the status of the host where the virtual machine resides.
2. If the status of the host is abnormal, the HA-enabled virtual machine will attempt to restart automatically.

1.6.1.4 IP/MAC/ARP Spoofing Protection

IP, MAC, and ARP spoofing can disrupt a network and allow attackers to intercept sensitive traffic.

The system isolates abnormal protocol access initiated by virtual machines at the host's data link layer and blocks virtual machine MAC/ARP spoofing. It also prevents virtual machine IP spoofing at the host's network layer.

1.6.1.5 Images and Snapshots

Images

You can create images from virtual machines. An image contains a complete data information of a virtual machine. You can use images to quickly replicate corresponding resources.

NexaSphere provides protection for image integrity and security:

- Images use encryption algorithms to protect integrity. When an image is downloaded from the image storage to a data storage, it must pass encryption algorithm verification. The download only proceeds if the verification is successful.
- Image files are stored in slices in the image storage. The sliced image files must be reassembled by NexaSphere before you can read their specific content, thereby ensuring the image data security.

Snapshots

You can create snapshots for virtual machines. A snapshot is essentially a data state file of a disk at a specific time. Before performing important operations, creating a snapshot for a virtual machine can retain the data state (including the memory state) at that specific time, facilitating quick rollback in case of failures. For long-term backup, it is recommended that you use the backup service.

The snapshot feature is applied in the following scenarios:

- Quick Failure Recovery: If an unexpected failure occurs in the production environment, you can use snapshot rollback to return quickly to a known state.

- Data Development: By creating snapshots of production data, you can acquire near real-time authentic production data for applications such as data mining, report query, and development testing.
- Improve Operation Fault Tolerance: Before major operations such as system upgrades or business data migration, we recommend that you create one or more snapshots. If any problem occurs during the upgrade or migration process, you can use snapshots to restore the normal system data state in time.

1.6.1.6 Encrypted Password Storage

All plaintext passwords in NexaSphere are stored in encrypted form to protect user data.

Supported scenarios for encrypted password storage include, but are not limited to:

- Host passwords: Not displayed in plaintext.
- Data storage passwords: Not displayed in plaintext.
- Database passwords: Encrypted with keys and not exposed to users.
- Task passwords: Passwords in all NexaSphere task logs are masked or hidden from users.

1.6.1.7 Resource Deletion Protection

Deletion Policy

Deletion policies protect important resources and reduce the risk of accidental deletion.

The current deletion policies include Immediate Deletion, Delayed Deletion, and Never Delete.

- Immediate Deletion: Resources are physically deleted directly and removed from the database . Deleted resources cannot be recovered.
- Delayed Deletion: A resource is marked as deleted in the database but is not physically removed immediately. Before the retention period expires, you can recover the resource from the recycle bin in the UI or by using the NexaSphere API. During this period, the resource continues to occupy physical capacity, such as disk space. After the retention period expires, the resource is physically deleted and cannot be recovered.
- Never Delete: Resources are marked as deleted in the database but are never physically deleted. They occupy physical space all the time.

Resources that currently support deletion policy include virtual machines, disks, images, and bare metal instances.

- VM Deletion Policy: Immediate Deletion, Delayed Deletion, and Never Delete. The default policy is Delayed Deletion.
- Disk Deletion Policy: Immediate Deletion, Delayed Deletion, and Never Delete. The default policy is Delayed Deletion.
- Image Deletion Policy: Immediate Deletion, Delayed Deletion, and Never Delete. The default policy is Delayed Deletion.
- Bare Metal Instance Deletion Policy: Immediate Deletion and Never Delete. The default policy is Never Delete.

UI Deletion Reminder

The UI provides a protection mechanism for deleting important resources. The system displays the consequences of deleting the resource and shows the number of directly associated virtual machines and disks. You must confirm the deletion to proceed, reducing the risk of accidental operation.

1.6.1.8 Monitoring and Alarms

Monitoring and alarms are provided through the monitoring and notification systems. The monitoring system tracks time-series data and events, while the notification system sends alarm messages to configured endpoints.

The monitoring system presents system performance, resource utilization, and other metrics through monitoring screens, dashboards, charts, and banner notifications. These views show NexaSphere resource utilization, operating status, and health. You can also configure alarms and endpoints for fine-grained monitoring and prompt diagnosis of issues.

Characteristics of the monitoring system:

- Time-Series Monitoring: The system currently supports monitoring two types of time-series data.
  - Resource Utilization Data: For example, VM CPU utilization and host memory utilization.
  - Resource Capacity Data: For example, the number of available IP addresses and the total number of running virtual machines.
- Event Collection: Collects predefined events that occur in NexaSphere, such as host disconnection and virtual machine high availability activation.
- Alarms: Generates alarms for time-series data or events and provides global notifications for important resources, such as the available physical capacity of data storage.

- Events: Records all operations and supports search.
- Customization: Allows you to customize alarms and alert message templates.

Characteristics of the notification system:

- Pushes alarm messages to specified endpoints.
- The system provides a system endpoint by default. You can set other types of endpoints.

1.6.2 Network Security

1.6.2.1 Security Group

Security groups filter TCP, UDP, ICMP, and other traffic for VMs and enforce configured security rules on specified NICs.

1.6.3 Access Control Security

1.6.3.1 Two-Factor Authentication

Two-factor authentication (2FA) adds a second layer of protection to static passwords. When 2FA is enabled in NexaSphere, each login requires the six-digit dynamic security code generated by the authenticator app.

After 2FA is enabled and you successfully log in for the first time, the login QR code is no longer displayed. This helps prevent malicious login attempts and further enhances system security.

1.6.3.2 AccessKey Authentication

AccessKey authentication is supported.

A local AccessKey consists of an AccessKey ID and AccessKey Secret. It is a security credential issued by NexaSphere that authorizes a third-party user to call NexaSphere APIs and access NexaSphere resources. Keep both values confidential.

1.6.3.3 Tasks and Events

Unified task logs record user logins and resource operations performed through all account types in NexaSphere. Each record includes the operation description, task result, operator, client IP address, task creation and completion times, and returned details. These task and event records support security analysis, intrusion detection, resource-change tracking, and compliance audits.

1.7 Openness and Compatibility

1.7.1 Open APIs

NexaSphere provides RESTful APIs and Java and Python SDKs for integration and automation. Developers can use standard HTTP methods and REST principles or call APIs programmatically through an SDK to implement the required functions.

The API Reference describes how to use the RESTful APIs and SDKs and provides definitions for the available APIs. The visual API Inspector in the UI displays invocation details for query and operation APIs. Developers can copy an API call and run it in a third-party debugging tool, simplifying API development and troubleshooting.

1.7.2 Hardware Compatibility

NexaSphere decouples software deployment from specific hardware platforms. You can select an appropriate hardware configuration and NexaSphere software components according to project requirements. NexaSphere does not require a particular server brand. If existing servers are to be reused, NexaSphere can incorporate compatible hardware and manage it uniformly through NexaSphere.

Glossary

2.1.1.1 Glossary

Data Center
A data center is the largest resource namespace within a virtualization platform, including resources such as clusters, hosts, data storage, distributed switches, and distributed port groups.

Cluster
A logical collection of a group of hosts (compute nodes).

Host
A host is an x86 or ARM physical server running a KVM virtualization hypervisor, providing resources such as computing, networking, and storage to virtual machines.

Virtual Machine Group
A logical grouping of virtual machines based on business needs.

Virtual Machine
A virtual machine is a virtualized host running on a physical host, capable of running an operation system and applications just like a physical host.

Disk
A disk provides storage space for a virtual machine. Disks are categorized into system disks and data disks.

System Disk
A system disk provides support for the system operations of a virtual machine.

Data Disk
A data disk provides extended storage space for a virtual machine.

Image
An image is a template file used by virtual machines or disks. Images are categorized into system images and disk images.

Image Storage
An image storage is a virtualized resource that provides storage space for image template files used by virtual machines or disks. An image storage can be categorized into standalone image storage and distributed image storage.

Standalone Image Storage
A standalone image storage stores image files through image slices and support incremental storage.

Distributed Image Storage
A distributed image storage stores image files through distributed block storage.

Data Storage
A data storage is a virtualized resource that provides storage space for virtual machines and their application data. A data storage can be categorized into local storage and network shared storage .

Local Storage
A local storage is storage resource constructed using the physical storage space of one or more hosts.

Netowrk Shared Storage
A storage system used for remote storage of virtual machines and their application data, accessible concurrently by hosts over a network.

Distributed Switch
A virtual switching device that provides unified virtual network management and monitoring for virtual machines within a cluster.

Distributed Port Group
A logical grouping of ports on a distributed switch, used for port configuration.

vNUMA Configuration

vNUMA uses CPU pinning to passthrough the topology of associated host physical NUMA (pNUMA) nodes to a virtual machine, generating a topology of virtual NUMA (vNUMA) nodes for the virtual machine. This topology enables a vCPU on a vNUMA node to primarily access the local memory and thus improves VM performance.

NUMA (Non-Uniform Memory Access)

Non-uniform memory access (NUMA) is a computer memory design where the memory access time depends on the memory location relative to the CPU. Under NUMA, a processor can access its own local memory faster than non-local memory and thus improves VM performance.

pNUMA Node (physical NUMA Node)

A pNUMA node (physical NUMA node) is a host NUMA node predefined based on the host NUMA architecture. It is used to manage the CPUs and memory of the host.

pNUMA Topology (physical NUMA Topology)

A pNUMA topology (physical NUMA topology) is the topology of the host NUMA nodes predefined by the CPU vendor based on the host NUMA architecture.

vNUMA Node (virtual NUMA Node)

A vNUMA node (virtual NUMA node) is generated by passing-through associated pNUMA nodes via CPU pinning. It is used to manage the CPUs and memory of a virtual machine.

vNUMA Topology (virtual NUMA Topology)

A vNUMA topology (virtual NUMA topology) is the topology of VM NUMA nodes generated by passing-through associated pNUMA nodes via CPU pinning.

Local Memory

Local memory is the memory that a CPU (pCPU or vCPU) accesses through the Uncore iMC (Integrated Memory Controller) of the same NUMA (pNUMA or vNUMA) node. Compared with accessing non-local memory, accessing local memory has lower latencies.

CPU Pinning

CPU pinning assigns the virtual CPUs (vCPUs) of a virtual machine to specific physical CPUs (pCPUs) of the host, which improves VM performance.

EmulatorPin Configuration

EmulatorPin assigns all other threads than virtual CPU (vCPU) threads and IO threads of a virtual machine to physical CPUs (pCPUs) of the host so that these threads run on assigned pCPUs.

Snapshot

A snapshot is a point-in-time capture of data status in a disk.

VM Scheduling Policy

A VM scheduling policy is a resource orchestration policy based on which virtual machines are assigned to hosts to achieve the high performance and high availability of businesses.

Management Node

A physical host where the system is installed, providing UI management and virtualization platform deployment capabilities.

Compute Node

Also known as a host, it is a physical server that provides computing, networking, and storage resources for virtual machines.

iSCSI Storage

iSCSI storage is an SAN storage that uses the iSCSI protocol for data transmission. You can add an iSCSI SAN block as a SAN storage or passthrough the block to a virtual machine.

FC Storage

FC storage is an SAN storage that uses the FC technology for data transmission. You can add an FC SAN block as a SAN storage or passthrough the block to a virtual machine.

NVMe Storage

A type of storage implemented via the NVMe-oF (NVMe over fabrics) protocol. You can add a block device configured from an NVMe storage as SAN storage.

Management Network

A management network is used to manage physical resources in the platform. For example, you can create a management network to manage access to hosts, data storage, and image storage.

Security Group

A security group provides security control services for VM NICs. It filters the ingress or egress TCP, UDP, and ICMP packets of VM NICs based on the specified security rules.

MN Monitoring

Management Node (MN) monitoring allows you to view the health status of each management node when you use multiple management nodes to achieve high availability.

Alarm

An alarm is used to monitor the status of time-series data and events and respond to the status change. Alarms can be categorized into resource alarm and event alarm.

Message Template

A message template specifies the text template of a resource alarm message or event alarm message sent to an SNS system.

Endpoint

An endpoint is a method that users obtain subscribed messages.

Alarm Message

An alarm message is a message sent the time when an alarm is triggered.

Operation Task

An operation task is a chronological record of operations on the specified objects and their operation results.

HA Task

HA task logs generated when the platform executes high availability procedures in accordance with the enabled HA policy.

Scheduling Task

Scheduling task logs generated when the platform executes dynamic resource scheduling operations after the cluster DRS is enabled.

Event

Event monitors and records all activities on the platform. You can use this feature to implement operation tracking, security analysis, troubleshooting, and automatic O&M.

Backup Plan

You can create a backup plan to back up local virtual machines or platform databases to a specified backup storage on a regular basis.

Local Backup Data

Local backup data of virtual machines or platform databases is stored in the local backup storage.

Local Backup Storage

A local backup storage is located at the local data center and is used to store local backup data.

Remote Backup Storage

A remote backup storage is located at the a remote data center and is used to store remote backup data.

Tag

A tag is used to mark resources. You can use a tag to search for and aggregate resources.

Custom Attribute

Custom attributes are user-defined key-value pairs in the platform that extend the resource tagging and management capabilities.

Single Sign-On

The Single Sign-On service supports seamless access to SSO systems. Through the service, related users can directly log in to the platform and manage resources.

Console Proxy

Console proxy allows you to log in to a virtual machine by using the IP address of a proxy.

AccessKey Management

An AccessKey pair is a security credential that one party authorizes another party to call API operations and access its resources in the platform. AccessKey pairs shall be kept confidential.

IP Blocklist/Allowlist

An IP blocklist or allowlist identifies and filters IP addresses that access the platform. You can create an IP allowlist or blocklist to improve access control of the platform.

User Management

A user can be created by the admin or synchronized from an authentication system. A user is managed by the admin. Resources created by a user are managed by the user.

Theme and Appearance

You can customize the theme and appearance of the platform.

Email Server

If you select Email as the endpoint of an alarm, you need to set an email server. Then alarm messages are sent to the email server.

Log Server

A log server receives management node logs or platform operation logs sent from the virtualiza tion platform for centralized retention, search, and analysis.

System Parameters

System parameters allow you to configure settings that take effect on the whole platform.

HA Policy

HA Policy is a mechanism that ensures sustained and stable running of the business if virtual machine are unexpectedly stopped or are errored because of errors occurring to compute, network, or storage resources associated with the virtual machines. By enabling this feature, you can customize VM HA policies to ensure your business continuity and stability.

Bare Metal Template

With bare metal templates, preconfigured files can be quickly generated to achieve unattended bulk OS installation for bare metal instances.

Bare Metal Cluster

A bare metal cluster consists of bare metal chassis.

Deployment Server

An independent server used for providing PXE services and console proxies for bare metal chassis.

Bare Metal Chassis

Bare metal chassis is used to create bare metal instances and can be universally identified by the BMC interface and IPMI configuration.

Bare Metal Instance

A virtualized instance of bare metal chassis.