OpenShift Virtualization Example Architecture: 1000 virtual machines

Hybrid Platforms Business Unit

version: 1.0.0

December 2025

Legal Notice

Table of Contents

About this Document

Find the latest version of this document at https://red.ht/virtrefguide.

Purpose of this document

The purpose of this document is to describe an example implementation of OpenShift Virtualization that illustrates how virtual machine workloads can be designed, deployed, and managed within an OpenShift environment. It provides architectural guidance and design considerations intended to help organizations plan a scalable, secure, and performant virtualization platform that aligns with enterprise standards and operational practices.

This document serves as a reference for enterprise and virtualization architects, platform engineers, and cluster administrators who are responsible for designing and implementing OpenShift Virtualization. Its goal is to outline a representative architecture, including compute, storage, networking, and availability design patterns, without prescribing detailed configuration commands or environment-specific procedures as those will be specific to the set of vendors used for compute, network, and storage services.

By presenting a structured, narrative-based example, the document enables readers to understand the rationale behind architectural decisions, how requirements and constraints shape the design, and how OpenShift Virtualization integrates with existing enterprise infrastructure. It focuses on Day 1 activities, covering the planning and implementation considerations necessary to establish a production-ready virtualization environment, while excluding Day 2 operational procedures such as maintenance, scaling, or upgrades.

Assumptions

This architecture guide assumes that the reader has the following knowledge:

- Foundational knowledge of OpenShift: The reader is familiar with the architecture and operational concepts of Red Hat OpenShift Container Platform, including cluster components, Operators, and general administrative functions.
- Understanding of enterprise infrastructure: The reader has experience with data center infrastructure concepts such as networking, storage, and compute virtualization, and understands how these components interact in an enterprise environment.
- Experience with virtualization technologies: The reader has prior exposure to traditional hypervisor-based virtualization platforms (such as VMware vSphere, Red Hat

Virtualization, or Hyper-V) and understands basic concepts like virtual CPUs, memory allocation, and disk provisioning.

- Awareness of security and compliance practices: The reader understands enterprise security fundamentals, including role-based access control (RBAC), network segmentation, encryption, and compliance standards relevant to infrastructure design.
- Access to enterprise-grade infrastructure: The implementation assumes the availability of physical or virtual hardware that meets OpenShift's system requirements, along with reliable storage and networking infrastructure capable of supporting virtualization workloads.
- Knowledge of Linux system administration: The reader is comfortable with Linux operating system concepts, shell-based management, and system-level configuration, as these skills are essential for deploying and maintaining OpenShift nodes.
- Familiarity with containerization concepts: The reader understands how containers and Kubernetes resources are orchestrated within OpenShift and recognizes how virtual machines are represented as workloads within the same ecosystem.
- Access to product documentation and support resources: It is assumed that the reader can reference official Red Hat documentation, subscription content, and support channels for detailed configuration or troubleshooting guidance beyond what this document provides.

To learn more about OpenShift and OpenShift Virtualization, Red Hat Learning Services offers multiple resources, including a guided learning path, the learning hub, and instructor-led courses, such as DO180, DO280, DO316, DO322, and DO380.

Target audience

This document is intended for enterprise and virtualization architects, platform engineers, and cluster administrators responsible for designing and implementing OpenShift Virtualization environments. Readers are expected to have a foundational understanding of OpenShift concepts, including cluster architecture, networking, and storage, as well as familiarity with traditional virtualization technologies and infrastructure operations.

The content assumes that readers are engaged in Day 1 planning and design activities, such as defining requirements, selecting hardware configurations, establishing network and storage topology, and determining high-availability and security strategies. It is particularly useful for those developing reference architectures, designing production environments, or integrating virtualization capabilities into existing OpenShift deployments.

While this document provides architectural context and design patterns, it is not intended for readers seeking detailed installation procedures or operational runbooks. Instead, it serves as a guide for technical decision-makers and practitioners who need to understand the architectural foundations and design rationale behind an enterprise-grade OpenShift Virtualization implementation. Specific installation procedures and runbooks can be found in partner-led reference architectures, such as those from Cisco (e.g. FlexPod, FlashStack), Dell, HPE, and other partners linked from https://red.ht/workswithvirt.

Scope and Boundaries

In Scope

The topics in this section represent the areas directly addressed by this example implementation. Each reflects a design-time focus on architecture, configuration intent, and platform readiness.

- Architectural overview and design intent: The document provides a conceptual and logical overview of OpenShift Virtualization, describing how virtual machine workloads are hosted within an OpenShift cluster. It outlines architectural rationale, value propositions, and integration of virtualization capabilities into a container-native ecosystem.
- Requirements, constraints, and assumptions: The guide defines business, technical, and operational requirements that shape the architecture, along with environmental constraints and assumptions. These inputs form the basis for design decisions and expected platform behavior.
- Physical and logical infrastructure design: Hardware and topology guidance is included for the OpenShift cluster, covering control plane, worker, and infrastructure node roles. It also describes physical network connectivity, rack placement, and storage considerations required to support virtualization workloads.
- Cluster-level configuration considerations: Key Day 1 configuration decisions, such as control plane settings and required OpenShift Operators and their configuration, are addressed to ensure the cluster is correctly prepared for virtualization workloads. This focuses on what must be defined during setup, not explicitly how to execute the installation.
- Node configuration and host-level tuning: The document details host-level settings that influence virtualization performance and reliability, including CPU and memory tuning, network and storage connectivity, and other relevant platform capabilities. These configurations ensure a consistent foundation for predictable VM operations.
- Virtual machine design and resource consumption: Configuration patterns for VM storage, networking, and compute are described to illustrate how virtual machines consume cluster resources.
- High-availability architecture and remediation design: The document defines strategies for achieving platform resilience, including node health checks, remediation mechanisms, and redundancy design for critical components. It focuses on architectural principles rather than detailed recovery runbooks.

- Security and compliance architecture: Security is addressed from an architectural standpoint through RBAC, security contexts, and compliance frameworks. The goal is to ensure that virtualization workloads maintain consistent enterprise security posture within the OpenShift environment.
- Observability and operational readiness design: Guidance is included for metrics, monitoring, and logging design, describing how these components provide visibility into virtualization health and performance. The emphasis is on architectural integration rather than troubleshooting processes.
- Foundational integration points: Core dependencies such as identity management, networking, and storage systems are discussed insofar as they influence virtualization architecture. These integrations are addressed conceptually, without delving into environment-specific configuration.

Out of Scope

To maintain focus on architecture-level design, several topics are intentionally excluded from this document. These subjects fall outside Day 1 implementation planning or vary significantly by organization and environment.

- Installation procedures and tooling: This document does not describe the specific methods or tools used to install OpenShift or OpenShift Virtualization, such as IPI, UPI, or the Assisted Installer. Step-by-step commands, manifests, and automation workflows are outside the scope of this example implementation, which focuses on design-level decisions rather than implementation execution.
- Day 2 operations and lifecycle management: Post-installation administrative activities, including upgrades, patching, scaling, and ongoing capacity management, are not covered. These are considered operational tasks rather than Day 1 architectural considerations and should be documented in operations or platform runbooks.
- Application and workload configuration: Configuration of guest operating systems, application deployments, or workload lifecycle processes within virtual machines is excluded. This guide focuses on the underlying virtualization platform, not the applications or services that run on top of it.
- Environment-specific customization: Infrastructure integrations unique to a particular organization, such as proprietary DNS, CMDB, ITSM, or security systems, are out of scope. The example implementation is intended to be environment-agnostic, emphasizing reusable architectural patterns rather than custom integrations.
- Cloud provider–specific and third-party integrations: Cloud-native or vendor-specific components, such as AWS, Azure, or GCP service integrations, as well as detailed

VMware migration tooling, are not included. The example architecture is designed to remain platform-neutral and applicable to on-premises or hybrid environments.

- Performance benchmarking and tuning results: The document references performance targets and sizing guidelines but does not include measured benchmark results or detailed tuning procedures. Performance validation is environment and workload-dependent and outside the architectural intent of this guide.
- Disaster recovery execution and backup runbooks: While high availability and recovery architecture are discussed, detailed DR procedures, such as step-by-step failover and restore operations, are out of scope. Those topics belong in operational recovery documentation rather than architectural design.
- Licensing and cost modeling: Pricing structures, subscription models, and cost analysis for OpenShift, virtualization, or infrastructure components are not included. These subjects vary by organization and procurement strategy and are not part of architecture design guidance.
- Precise implementation and configuration of security features and capabilities: OpenShift's security features are broad and encompass all aspects of the platform and virtualization, from the hypervisor operating system to the virtual machine runtime environment to administrative controls on utilization. This document does not include security configuration guidance or suggestions, however the OpenShift Virtualization Hardening Guide should act as a starting point with additional use of the OpenShift Compliance Operator's profiles for further security feature configuration.
- Security operations and incident response: Security architecture is covered conceptually through design elements such as RBAC, SCC/PSa, and encryption, but operational security functions, like incident handling, vulnerability scanning, and penetration testing, are not addressed. These activities are part of ongoing enterprise security operations.
- Experimental and unsupported features: Configuration, components, and capabilities classified as Tech Preview, Developer Preview, or unsupported in production are intentionally excluded. This document focuses on generally available, production-supported features to ensure design stability and reproducibility.

Overview

Technology Overview

Red Hat OpenShift is an enterprise-grade Kubernetes platform that provides a consistent, secure, and automated foundation for running modern applications across on-premises and cloud infrastructure. It integrates core Kubernetes capabilities with opinionated enhancements, including Operators, declarative lifecycle management, built-in observability, and robust security controls, to create a fully supported platform for both developers and operators. OpenShift's architecture unifies networking, storage, compute, and cluster services under a standardized operational model, enabling organizations to deliver applications more reliably while maintaining compliance, scalability, and operational consistency.

OpenShift Virtualization extends this platform by enabling traditional virtual machine (VM) workloads to run natively alongside containerized applications. Built on the KubeVirt project and backed by the proven, high-performance KVM hypervisor, it brings VMs onto the platform where they benefit from the same scheduling, networking, storage, automation, and security frameworks that OpenShift provides. This unification allows organizations to consolidate their environments, simplify infrastructure management, and support modernization initiatives by gradually transforming or integrating legacy applications with cloud-native architectures, all within a single, cohesive platform.

Architectural Requirements

This section defines the business, technical, and operational objectives that the OpenShift Virtualization architecture must satisfy. These requirements outline what the environment is expected to deliver, such as workload capacity, performance targets, availability goals, and integration needs, which serve as the foundation for all subsequent design decisions.

- Compute capacity
    - Must support 1,000 VMs averaging 4 vCPU and 16 GiB RAM.
    - CPU overcommit of 4:1 and memory overcommit of 2:1 will be used for capacity planning purposes.
- Storage
    - Virtual machine storage is provided using an external storage array using iSCSI.
        - Storage supports VM disk snapshot and clone operations.
- Network

    - VM / Application traffic physically isolated from other traffic types
    - Support multiple VLANs used for traffic separation for both cluster traffic (management, storage, migration) and workload (VM interfaces).
- Management
    - The platform must support live migration for all eligible workloads, including database and similar workloads with high memory requirements.
    - Authentication via external LDAP-based provider.
- High availability
    - In the event of node failure, virtual machines should (re)start in 120 seconds or less.
    - Minimum of 2 server or 10%, whichever is greater, failure capacity.

Design Constraints

Design constraints describe the fixed conditions and limitations that the architecture must operate within, such as existing infrastructure, security policies, hardware standards, or organizational requirements. These constraints differ from requirements by defining what cannot be changed and therefore directly influence which design choices are feasible.

- Only fully supported, generally available features may be used in production.

Additional Assumptions

These additional assumptions have been integrated with the architecture implementation described in this document.

- Hardware virtualization support
    - All compute nodes are assumed to support hardware-assisted virtualization (Intel VT-x/AMD-V) and IOMMU, ensuring compatibility with KubeVirt and features such as SR-IOV and CPU pinning where applicable. Additionally, it is assumed that these features have been enabled in the hardware prior to deployment and configuration.
- Enterprise network availability
    - The datacenter network provides reliable connectivity between racks, switches, and supporting services (DNS, DHCP or IPAM, NTP, PKI). Appropriate VLANs and routing paths exist or can be provisioned to support workload, storage, and migration traffic.
- Storage platform preparedness

    - The storage array or CSI-backed storage system is assumed to be correctly zoned, presented, and accessible to the OpenShift nodes. Any required LUN mapping or array-side configuration outside of OpenShift is completed by storage administrators.
- Identity and access integration
    - The corporate identity provider integration is assumed to be available and functional, enabling platform-level RBAC policies and multi-team access patterns consistent with enterprise standards.
- External cloud dependencies
    - This architecture relies on an internet connection for downloading container images used by OpenShift features and capabilities. No local mirror configuration is described in this document.
- Access to product documentation
    - Administrators and architects working with this example implementation are expected to reference official Red Hat documentation for detailed configuration steps, Operator settings, and troubleshooting guidance outside this architectural scope.

Datacenter and hardware

This section defines the datacenter prerequisites and physical hardware profile required to successfully implement this OpenShift Virtualization architecture. It documents the assumptions and design decisions made at the infrastructure layer, covering compute, network, and storage. The configuration documented provides clear expectations for the underlying datacenter capabilities needed to achieve the scale, performance, and availability characteristics described throughout this architecture.

External dependency assumptions

An OpenShift Virtualization deployment depends on established enterprise services to provide some functionality. This section outlines the assumptions about the enterprise environment that have been made in this architecture. These assumptions do not prescribe how services must be implemented. Rather, they clarify the baseline environmental capabilities expected to be available before cluster installation and hardware configuration begin.

- Internet connected cluster (no local mirror) - The cluster is assumed to have outbound internet access to retrieve container images, Operator content, and platform updates. Because no local image mirror or disconnected registry is planned, all cluster components rely on upstream Red Hat sources for installation and ongoing lifecycle operations.
- External network services are available, correctly configured, and reliable - Core datacenter services such as DNS, load balancing, NTP, and identity providers are assumed to exist and operate at levels of reliability consistent with enterprise production environments.
    - DNS and load balancers for API and *.apps: The OpenShift API and Ingress endpoints must resolve correctly and be reachable through appropriately configured load balancers.
    - NTP: Time synchronization is required across all nodes to ensure consistent behavior of distributed systems such as etcd, authentication, and logging.
    - Authentication systems: The environment must provide a functioning identity service capable of integrating with OpenShift, enabling user and group-based RBAC models.
- Static IP vs DHCP is out of scope - The method by which node addresses are assigned is not constrained by this design. Both static IP assignments and DHCP-based provisioning are valid, and the architecture does not introduce requirements that favor one method. It is assumed the organization will follow its existing IPAM practices.

- Authentication provider is out of scope - While the existence of a functional authentication system is assumed, the specific configuration of the external identity provider (such as LDAP schemas, filters, or mappings) is beyond the scope of this document. The architecture only relies on the presence of a working enterprise authentication mechanism rather than defining how it must be implemented.

Enterprise storage

This architecture assumes the use of an enterprise-class external storage array that is capable of delivering the aggregate capacity, IOPS, and latency characteristics required to support high-density virtual machine workloads at scale. The storage platform must be able to sustain consistent performance under mixed workload conditions while also supporting the volume count and concurrency expected from hundreds to thousands of virtual machine disks. It is further assumed that the storage vendor provides a CSI driver that is fully supported with both OpenShift and OpenShift Virtualization, enabling the definition of StorageClasses that explicitly describe the capabilities required by applications and virtual machines, such as performance tiers, access modes, and availability characteristics. Support for persistent volume snapshots and cloning is also required, as these capabilities are foundational to common virtualization workflows including VM provisioning, templating, backup, and recovery, and are treated as baseline expectations rather than optional enhancements in this design.

ReadWriteMany (RWX) persistent volumes are required for virtual machine live migration and any storage solution must include this capability for all volumes, regardless of block or filesystem mode.

Storage from NetApp, Dell, Pure Storage, IBM, and Hitachi, among others, are known to be capable of meeting these requirements. See https://red.ht/workswithvirt for the list of certified storage integrations with OpenShift Virtualization. Identifying and choosing a specific storage vendor and enterprise array are outside the scope of this document.

Datacenter networking

This architecture assumes an enterprise-grade datacenter network infrastructure that is capable of supporting the connectivity, performance, and resiliency requirements of a large-scale virtualization deployment. The physical switching fabric must provide sufficient port availability, bandwidth, and forwarding capacity to accommodate dense compute nodes, multiple bonded interfaces, and the traffic patterns introduced by virtual machine workloads, storage access, and live migration. Low and predictable latency, non-blocking throughput, and redundancy at both the switch and link layers are treated as baseline expectations, ensuring that network performance does not become a limiting factor for virtual machine density, migration reliability,

or overall platform availability. These characteristics are considered prerequisites for implementing the node- and workload-level network configurations defined later in this architecture.

Node hardware configuration

The hardware configuration defines the physical foundation on which the OpenShift Virtualization platform is deployed and operates. This section translates the architectural requirements, such as workload density, overcommit ratios, high-availability expectations, and operational resiliency, into concrete hardware specifications that support predictable and scalable platform behavior.

This section defines the opinionated hardware characteristics required for compute, memory, storage, and network subsystems that meet the requirements defined. The intent is not to explore a range of possibilities but to specify a configuration that is capable of delivering reliable results for clusters of this scale. By examining CPU requirements, memory footprint, local storage needs, and network connectivity, the following guidance establishes the baseline for a consistent and repeatable hardware design that supports both current and future virtualization needs.

Compute

The compute layer for this OpenShift Virtualization environment must be sized to guarantee predictable performance, support the required virtual machine density, and maintain sufficient headroom for node failures and maintenance events. Given the characteristics of the target workload and the architectural posture of this implementation, the following requirements and design decisions define the basis for all sizing calculations:

- 1,000 virtual machines, each expected to run continuously under typical enterprise load.
- Average VM profile: 4 vCPUs and 16 GiB of memory.
- CPU overcommit ratio: 4:1, appropriate for mixed workloads with moderate CPU utilization.
- Memory overcommit ratio: 2:1, reflecting conservative memory pressure tolerance for VM workloads.
- Maximum VMs per node: 200, providing an upper boundary that limits operational risk, i.e. blast radius, during host failure scenarios.

Cluster architecture decisions:

- All nodes use a homogeneous hardware configuration, which simplifies scheduling, reduces variance in performance, and improves resiliency outcomes.
- Hyperthreading (SMT) is enabled, and each thread is counted as 0.5 physical cores to reflect real-world KVM scheduling behavior.
- A three node control plane will be used. This provides resiliency and high availability without the additional overhead and performance implications of a 5-node control plane. Additionally, should physical node failure occur and replacement will take an extended amount of time, since all nodes are homogenous then any surviving node can be converted into a control plane node to restore high availability.
- Control plane nodes are schedulable, ensuring full utilization of available compute resources without compromising platform reliability.
- No dedicated infrastructure nodes are required for this architecture, as planned platform components can be hosted efficiently across the cluster.
- Failover capacity is fixed at the greater of 2 nodes or 10% of total compute, establishing a clear, enforceable availability target.

Step-by-step compute sizing

With the information above, use the cluster sizing guide to determine the size and count of servers needed.

1. Determine the raw capacity required for workloads

   1000 VMs * 4 vCPU = 4000 vCPUs
   1000 VMs * 16 GiB memory = 16000 GiB memory, plus 292 GiB memory overhead

   The total resources needed for workload to 4000 vCPUs and 16292 GiB memory.

2. Factor overcommitment for CPU and memory

   4000 / 4 = 1000 CPU cores
   16292 / 2 = 8146 GiB of memory

   This shows the hardware capacity that we need to have to accommodate the workload. The average VM will consume 1 vCPU and 8.146 GiB of memory.

3. Identify node size

   The simplest way to determine node size is to work backwards by first identifying

acceptable node sizes. These can be based on organizational policies, such as whether single, dual, or quad socket servers are acceptable or can be done without constraints. Importantly, for each node size we must account for the node overhead (see step 3 in the sizing guide for how this is calculated) and deduct that from the amount of resources available to the workload.

Additionally, we want to use CPU:memory ratios that align with the virtual machine's requirements. For this environment, that is 1:8 so we will use node sizes which approximate that ratio.

Node CPU	Calculated node capacity[1]	Workload capacity	Cluster node count	# of VMs per host
16C / 32T	24 cores / 192GiB	23.8 cores / 181GiB	47	21
24C / 48T	32 cores / 256GiB	31.8 cores / 244 GiB	35	28
32C / 64T	48 cores / 384GiB	47.8 cores / 370GiB	23	43
48C / 96T	72 cores / 512GiB	71.7 cores / 495GiB	17	58
64C / 128T	96 cores / 768GiB	95.6 cores / 746GiB	12	83
96C / 192T	144 cores / 1024GiB	143.5 cores / 997GiB	9	111
128C / 256T	192 cores / 1536GiB	191.4 cores / 1499 GiB	6	166
192C / 384T	288 cores / 2048GiB	287 cores / 2000 GiB	5	200

[1] The architecture decisions, in the opening statement of this section, indicate that Hyperthreading / SMT is in use and threads count as .5 core for capacity purposes. The calculated node capacity is determined by using the formula `physical_core_count + ((thread_count - physical_core_count) * .5)`. For example, a server with 16 cores / 32 threads is calculated to have `16 + ((32 - 16) * .5) = 24` cores for sizing purposes below.

The requirements indicate that we can have no more than 200 VMs per node, so we stop when we reach that point. The requirements also indicate that we must account for

either 2 nodes or 10% of capacity as failover, whichever is greater. The following table adds this requirement into the cluster size, rounding up for fractional node count.

Node CPU	Calculated node capacity[1]	Workload node count	HA node count	Cluster node count
16C / 32T	24 cores / 192GiB	47	5	52
24C / 48T	32 cores / 256GiB	35	4	39
32C / 64T	48 cores / 384GiB	23	3	26
48C / 96T	72 cores / 512GiB	17	2	19
64C / 128T	**96 cores / 768GiB**	**12**	**2**	**14**
96C / 192T	144 cores / 1024GiB	9	2	11
128C / 256T	192 cores / 1536GiB	6	2	8
192C / 384T	288 cores / 2048GiB	5	2	7

[1] The architecture decisions, in the opening statement of this section, indicate that Hyperthreading / SMT is in use and threads count as .5 core for capacity purposes. The calculated node capacity is determined by using the formula `physical_core_count + ((thread_count - physical_core_count) * .5)`. For example, a server with 16 cores / 32 threads is calculated to have `16 + ((32 - 16) * .5) = 24` cores for sizing purposes below.

As a result, we see that as the node sizes increase beyond 96 cores / 768 GiB memory there are additional resources that will be unused as a result of the 2-node failover minimum, making this node size the ideal choice.

Our cluster will need 14 nodes that are 96 cores and 768 GiB memory to accommodate the workload and failover / HA capacity requirements. This will result in a maximum of 83 VMs per node, such as when failure has occurred and consumed the allocated HA capacity from the cluster.

4. Adjust for cluster architecture

As indicated above, a schedulable control plane will be used for this cluster. Using the control plane sizing recommendations, a cluster with fewer than 24 nodes needs 4 vCPU and 16GiB memory on each of the control plane nodes. To determine if this will affect our node count, calculate the amount of capacity used on each node for the workload.

Note: We remove the failover / HA capacity from this calculation to ensure that if that capacity is not available due to failure the cluster can continue to function as normal.

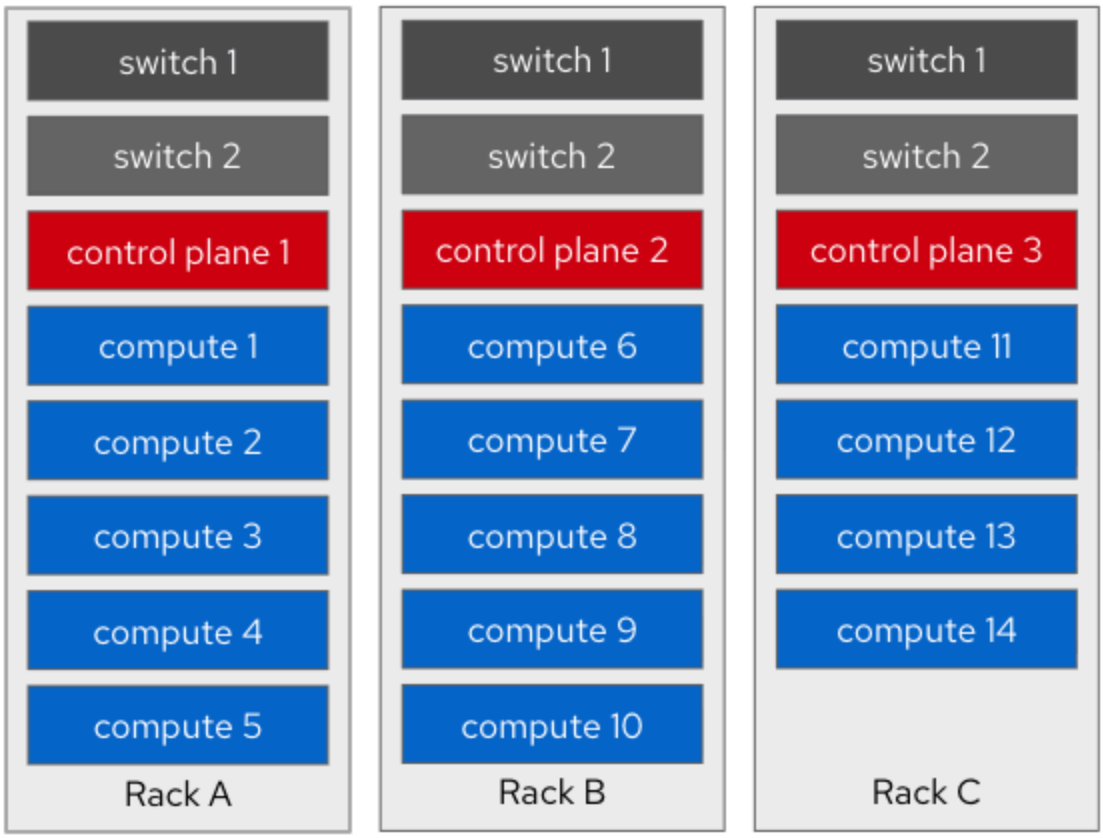
83 * 1 vCPU = 83 cores
83 * 8.146 GiB memory = 676 GiB memory

This leaves 13 cores and 92 GiB memory available capacity on the node, which exceeds the resources needed for control plane functions.

This cluster will not have dedicated infrastructure nodes, however no platform features which require significant additional resources are planned for this deployment.

The result is that our cluster will comprise 14 nodes, each with 96 cores and 768 GiB of memory.

Neither the requirements nor constraints for this architecture indicate any datacenter footprint / layout requirements or restrictions. However, if possible, it is ideal to stripe cluster members across racks in the datacenter to avoid an individual rack being a single point of failure for the entire cluster. The below diagram is an example, not a requirement, of a 3-rack deployment for the 14-node cluster described in this architecture. Ideally, the minimum number of racks used allows the control plane nodes to be distributed, no more than one per rack, to avoid additional effort to recover the control plane in the event of rack failure.



Network

The network architecture for this OpenShift Virtualization environment must provide deterministic throughput, fault tolerance, and strict traffic isolation to support high-density VM workloads. Because network bandwidth and resiliency directly influence live-migration performance, VM responsiveness, and overall cluster stability, the design mandates a clear separation of traffic classes and sufficient link capacity to prevent contention between storage, migration, and guest workload flows.

The following requirements form the basis for all network hardware decisions in this implementation:

Network Requirements Influencing Hardware Design

- Physical isolation of application (VM) traffic from all cluster infrastructure traffic, preventing noisy-neighbor effects and ensuring predictable workload performance.
- Fault tolerance at the NIC level, with tolerance of a single port failure for every traffic type, requiring duplication of links and bonds across independent adapters.
- Support for multiple VLANs to segment management, storage, migration, and workload networks according to enterprise policy and OpenShift best practices.

Storage-Driven Network Requirements

- External CSI-provisioned VM storage over iSCSI, requiring high-throughput, low-latency network paths with no dependence on link-aggregation for point-to-point performance.

Collectively, these requirements necessitate network interface configurations that deliver both sufficient aggregate bandwidth and true fault isolation, not merely logical separation. As a result, this design explicitly favors NIC configurations that offer high per-port throughput, predictable failover behavior, and the ability to dedicate links to workload-critical flows such as iSCSI and live migration.

Node network configuration

This indicates that we must have at least two physically separate network ports which can be used to separate application traffic from other types. Additionally, in order to tolerate single port failure we need to have at least two ports for each of those traffic types. This results in a minimum of four ports per node.

Link aggregation, such as a mode 4 (LACP, 802.3ad) bond of 2+ NICs, is commonly used to increase the total throughput available to a server. This works by distributing connections across the available network adapters using a hashing algorithm, typically based on factors such as the source and destination MAC addresses. This is effective when there are one:many connections between source and destination, such as a VM with a single NIC connecting to many clients, where the algorithm can distribute the connections across the available members. However, this is not effective when high throughput is needed between two singular endpoints, such as node-to-node or node-to-storage. The algorithm will resolve the connection to the same link each time, resulting in the throughput of only a single NIC available for the traffic flow.

IP-based storage often has both high throughput and low latency requirements. Importantly, when throughput and latency for storage access is affected it has a ripple effect across many VMs. As a result, this traffic needs to be protected from noisy neighbors to the extent possible.

Live migration traffic has the potential to consume significant amounts of throughput, which means that it can be considered a noisy neighbor. However, not giving adequate throughout to live migration traffic can result in extended time for migrations to complete and/or failed migrations.

The final factor to consider before continuing is the density of virtual machines per node. At maximum, the servers are expected to host 83 virtual machines plus the expected OpenShift platform features and connectivity requirements. In the absence of known network requirements for the applications and guest operating system functions, we will assume a single

1 Gb NIC for each VM and use a 3:1 network oversubscription for application and guest OS traffic. This means we want an aggregate throughput of 83 Gbps / 3 = 28 Gbps for VM traffic.

Our final requirements for networking are summarized in this table:

Purpose	Throughput	More info
Management (SDN)	Low (< 1 Gbps)	
Live migration	Bursty (up to link speed)	Can be limited to prevent noisy neighbor
iSCSI	High (up to link speed)	Avoid limiting to prevent VM disruption. Link aggregation does not increase point-to-point throughput.
Virtual machine	Variable (up to 28 Gbps)	Must use separate links per requirements

Depending on the link speed available for the servers, we can arrive at different configurations:

- 1 GbE - not feasible for other than management / SDN traffic.
- 10 GbE
    - Suitable for live migration and management / SDN.
    - May not be suitable for iSCSI traffic as a result of link aggregation required to reach desired throughput.
    - Will require 3+ link aggregation for VM traffic.
- 25 GbE
    - Suitable for live migration and management / SDN.
    - Suitable for iSCSI traffic, providing significant single-NIC throughput.
    - Will require 2+ link aggregation for VM traffic.
- 50 GbE
    - Suitable for all traffic types.

Even though 50 GbE links are suitable for all traffic types, our requirements dictate that we have two separate sets of links to isolate VM traffic from other types and redundant links - a minimum of two - to accommodate link failure. The cost of 50 Gb networking to the host may or may not be a factor.

10 GbE is a valid option, with two important factors to consider. First, live migration and iSCSI traffic may compete for bandwidth and interfere with each other if using shared links. Second,

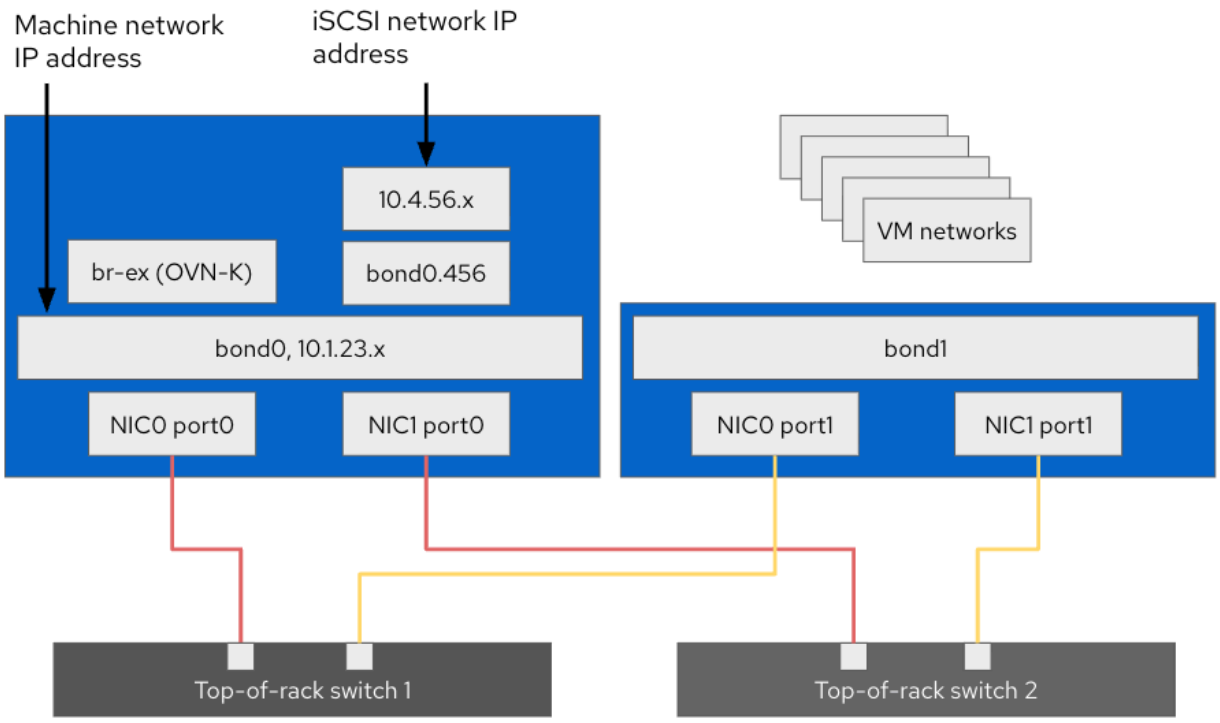
10Gb may not be adequate throughput for iSCSI traffic depending on the workload of the virtual machines.

As a result, the ideal configuration for this environment is to use 25GbE connectivity with a minimum for four ports across two NICs. Furthermore, to provide resiliency against either NIC or switch failure, the uplinks should be distributed across available upstream switches.

NIC	Port	Bond	Traffic type
0	0	bond0	Management / SDN, live migration, iSCSI
0	1	bond1	Virtual machine
1	0	bond0	Management / SDN, live migration, iSCSI
1	1	bond1	Virtual machine

Importantly, as a result of not using physically separate network adapters for the live migration network, a separate logical interface, i.e. VLAN, will not be used. Live migration traffic will utilize the SDN and, using configuration described later in this document, limit throughput to avoid starving the iSCSI network of bandwidth.

VLAN IDs and IP addresses are outside the scope and not specified in this document, however each node will require two IP addresses, one each on for the machine network and iSCSI network.



Storage

Local storage on each node is a critical component of the overall hardware design for this OpenShift Virtualization environment. Although virtual machine persistent storage is delivered through an external CSI-backed array, the platform still requires high-performance, low-latency local disks to support core OpenShift operations, virtualization runtime components, and memory overcommit features. Because these functions directly impact cluster stability and VM performance, node-local storage capacity and performance characteristics must be treated as architectural requirements—not adjustable parameters.

Cluster and Workload Requirements Influencing Storage Design

- High VM density per host, resulting in increased demand for image storage, ephemeral data, and logging capacity on each node.
- Memory overcommit ratio of 2:1, which necessitates fast, low-latency swap to protect VM performance during transient memory pressure events.
- No use of local storage for VM persistent volumes, ensuring that all node-local capacity is optimized for platform functions rather than guest OS data.

Platform and Control Plane Requirements

- Core RHCOS operating system requirements, including image storage, logs, and system metadata that must be hosted on resilient, sufficiently sized local disks.
- etcd performance constraints (on control plane nodes only), which require consistently low write latency to maintain cluster health.
- Container runtime and virtualization components, such as virt-launcher images and ephemeral VM scratch space, which rely on fast storage to avoid bottlenecks during VM lifecycle operations.

Collectively, these requirements determine the minimum capacity needed per node and the performance profile of the underlying devices.

Node storage configuration

The above platform and control plane requirements describe several different types of workload that will be hosted on the nodes. Each of these has different capacity requirements that must be taken into account:

Storage for	Estimated capacity	Additional details
Core RHCOS functionality	100 GiB	The minimum size for a RHCOS node disk is 100GiB and should be considered the amount of capacity needed for core RHCOS functionality. No specific performance requirements.
Container image caching	100-300 GiB	Used for container images, such as virt-launcher running on the node. May consume more if very large container images are used. No specific performance requirement, but slow storage can extend image pull times.
Container logs	10-20 GiB	Depends on the number of containers and the amount of logs they generate. No specific performance requirements.
Ephemeral storage for Pods	Variable	Depends on how many Pods using ephemeral storage, such as emptyDir, are running on the node. Expected to

		be low on a cluster dedicated to virtualization. No specific performance requirements, but will depend on what the ephemeral storage is being used for.
Swap for virtual machines	Up to 100% node memory	Capacity used depends on overcommit levels and feasibility of VM migrations if swap occurs. High performance, low latency storage is highly recommended.
etcd (control plane only)	8 GiB	etcd database is limited to 8 GiB. Low latency storage is required.

The documentation recommends that swap space capacity be determined by the amount of overcommit desired. For this architecture, 2:1 overcommit means there is 100% overcommit, which translates to 768GiB of swap capacity on the nodes.

Note: The desired memory overcommit ratio of 2:1 is modest and the overall cluster capacity is expected to be approximately 90%, even during failover events. This lowers the overall likelihood of swap events happening and, when they do, there is increased probability that another host will be available to migrate the VM in order to avoid swapping on an overused node. As a result, reducing the amount of capacity for swap may be deemed an acceptable risk by the business.

The last factor to accommodate when sizing storage capacity for the node is the eviction thresholds used by kubelet. There are two thresholds which are monitored by kubelet and will trigger workloads to be evicted if they are exceeded. They are:

- `nodefs.avaialble`, represents free capacity remaining on the filesystem where `/var/lib/kubelet` is. It defaults to 10%.
- `imagefs.available`, represents the free capacity remaining on the filesystem used for the container images, and when no "split image" configuration is used, the writable layers as well. With OpenShift, this defaults to `/var/lib/containers` and with no split image configuration used. The default value is 15%.

By default, and this architecture does not change the default, OpenShift uses the same disk and partition for both `/var/lib/kubelet` and `/var/lib/containers`. When more than one

filesystem type is hosted together, e.g. on the same partition, kubelet will use the larger value, which is 15% as specified by imagefs. This means that when the disk reaches 85% capacity, the system will treat it as full and will begin to evict workload. As a result, we must accommodate that free space requirement when capacity planning.

To determine the local storage capacity, sum the results from above, which comes to approximately 1 TiB of high throughput, low latency storage, and add the 15% required free space. Since both swap and etcd require the lowest latency possible, ideally this is flash-based NVMe storage. Write optimized storage is recommended, but not required, should swap events occur.

The result is that we need a minimum of 1.15 TiB of local storage capacity. To simplify and accommodate growth as well as unexpected local storage utilization, this is rounded up to 2TB.

Note: There are no requirements for this architecture for nodes to tolerate local storage failure, therefore having multiple disks in a software or hardware RAID configuration is optional, but not required. Additionally, there is no requirement for separate disks to be used for the root, swap, or containers (/var) partitions. These may be isolated for various cost, performance, and data protection needs as desired.

Cluster Deployment and Install

This section describes the configuration considerations that shape the cluster before, during, and immediately after deployment. Its purpose is to connect the architectural design decisions outlined earlier with the practical steps required to bring an OpenShift environment into an operational state suitable for virtualization workloads.

Configuration at deployment

Install method

The specific installation method used, i.e. Agent-based Installer, Installer-provisioned Infrastructure, User-provisioned Infrastructure, etc., is out of scope for this document. Any supported method may be used, provided it can reliably apply the required installation-time configurations defined in this document. Regardless of the method used, the below install-time configuration is applied.

Schedulable control plane

In order to optimize hardware utilization this cluster will not have dedicated control plane nor infrastructure nodes for OpenShift cluster services. As a result of the cluster being 14 nodes in size, the control plane does not need to be marked as schedulable during installation, however there is also no reason not to set it this way at install time.

Refer to the documentation for how to set control plane nodes to schedulable at install time. If this configuration is not applied at install, follow the documentation to change the control plane nodes to schedulable.

Cluster network MTU

By default the cluster will configure an MTU of 1500 for all network interfaces. There is nothing inherently wrong with this configuration, however when using jumbo frames it is ideal to configure them at install time. If needed, the MTU can be changed post-install. There are no requirements for a specific MTU value to be used, however if you are considering using hosted control planes with OpenShift Virtualization node pools at any point in the future, using jumbo frames (MTU larger than 1500) may improve compatibility for clusters using the hosting cluster's SDN for their primary network, which will result in double encapsulation.

The documentation describes how to set the MTU at install time.

Primary network interface configuration

As was decided in the node network configuration section, the host network will be configured to use two bonds, each one using two host interfaces. The bond that is the base layer for hosting OVN-Kubernetes, which is determined by the node associating the interface's IP address with the subnet assigned to the machine network, cannot be modified after node deployment, therefore it's important to ensure that its configuration is correct when deploying the node to avoid having to redeploy to adjust the configuration.

This document does not provide a prescriptive install method nor precise hardware configuration, as a result it is not possible to give exact network configuration guidance. However, an example NMstate snippet for bond0 and the iSCSI VLAN is provided below. Note that the choice of IP configuration (DHCP vs static) is outside the scope of this document. Additionally, the iSCSI VLAN interface can also be configured post-deployment if install-time configuration is not used.

```
    interfaces:
       # bond used for machine network (SDN, live migration)
       - name: bond0
         type: bond
         state: up
         mtu: 9000
         link-aggregation:
           mode: 802.3ad
           port:
             - enp0s0
             - enp1s0
         ipv4:
           enabled: true
           auto-dns: true
           auto-gateway: true
           auto-routes: true
           dhcp: true
         ipv6:
           enabled: false
         lldp:
           enabled: true
       # VLAN interface used for iSCSI traffic
       - name: bond0.456
         type: vlan
         state: up
         mtu: 9000
         vlan:
           base-iface: bond0
           id: 456
```

```
          ipv4:
            enabled: true
            address:
              - ip: 10.4.56.x
                prefix-length: 24
            auto-dns: false
            auto-gateway: false
            auto-routes: false
            dhcp: false
            enabled: true
          ipv6:
            enabled: false
```

Additional examples and full configuration options can be found in the documentation.

It is critical to ensure that the machine network CIDR is correctly defined in the install config when deploying the cluster so that the nodes will use the correct interface to instantiate and configure the SDN.

Workload partitioning

It is also recommended that since the control plane nodes are also being configured to allow virtual machine and workload placement, that workload partitioning be enabled and configured to ensure that a number of CPUs on each node are reserved for the control plane functions so that the nodes do not become overloaded and jeopardize the health of the cluster as a whole.

Post-deployment cluster configuration

Kubelet configuration

The kubelet is the primary agent responsible for managing workloads on each node, making its configuration significant to the behavior, performance, and reliability of virtual machine workloads in this environment. Because this architecture relies on high-density VM placement, memory overcommit, predictable eviction behavior, and strict control over node-level resources, the kubelet must be configured explicitly to ensure that node conditions are reported accurately and that the scheduler receives timely, actionable signals about resource pressure.

Kubelet's reporting, eviction, and resource-accounting parameters define how the platform interprets node health, how it responds to memory pressure, and when it initiates VM migration or cleanup. For high-density virtualization clusters, kubelet defaults may not reliably communicate resource pressure early enough, nor do they scale to the number of Pods and container images expected on nodes hosting hundreds of virtual machine workloads.

kubeAPIQPS and kubeAPIBurst

`kubeAPIQPS` and `kubeAPIBurst` are increased in this architecture to ensure that the kubelet can report node and workload state without API throttling, which is critical in a high-density virtualization environment where virt-launcher pods, live migrations, and remediation workflows generate frequent and sometimes bursty updates. Raising these limits allows the kubelet to sustain a higher baseline rate of API calls while also accommodating short spikes, reducing delays in status reporting and improving scheduler responsiveness during events such as VM placement, migration, and node health transitions.

Per the OpenShift Virtualization Tuning and Scaling Guide, a value of 100 for kubeAPIQPS and 200 for kubeAPIBurst is used.

maxPods

By default, OpenShift configures a maxPods threshold intended for typical containerized workloads. This architecture, however, relies on nodes hosting a large number of virtual machines, which represent one Pod per VM, plus supporting platform components. Increasing the value ensures that VM workloads are not artificially constrained by the pod limit and that the node's capacity aligns with the target VM density derived from sizing calculations.
Setting the value to 500, vs the default of 250, ensures kubelet allows creating enough Pods to accommodate all workload, including during failover, plus required overhead for control plane workloads on schedulable control plane nodes.

nodeStatusMaxImages

By default, the kubelet reports a size-sorted list of container images present on each node, limited by the nodeStatusMaxImages value, to prevent excessively large API objects. This list is consumed by the imageLocality scheduling plugin, which awards a score to nodes that already contain the images needed by a pending Pod. In high-density virtualization clusters, where nodes may store far more than the default 50 images, this scoring can unintentionally outweigh resource-based scheduling signals and lead to imbalanced VM placement. To avoid image locality overshadowing core scheduling priorities—such as CPU, memory, and topology—it is recommended to disable the nodeStatusMaxImages limit, ensuring that the presence of many container images does not distort the scheduler's scoring logic or bias placement toward heavily loaded nodes.

Set the value to -1 to disable the limit on the number of images reported to the control plane.

System resource reservation

By default, kubelet reserves a small, fixed amount of CPU and memory for host operations using the system-reserved setting, which may be insufficient on high-density virtualization nodes or systems with very large memory footprints where VM usage can approach total node capacity. Enabling autoSizingReserved ensures that reserved resources scale proportionally with total node memory, preventing system-level starvation during periods of heavy workload consumption.

To allow the system to automatically configure the amount of kubelet reserved resources based on the node's total resources, set the kubelet config value `autoSizingReserved` to `true`.

Note: reserving this capacity has already been factored into the total cluster sizing. See the cluster sizing guide for further details on how the values are calculated.

Soft Eviction Policies

Soft eviction thresholds define when the kubelet signals that a node is under resource pressure and begins gracefully evicting workloads. In a virtualization environment, this will result in the virtual machines being gracefully shutdown on the node and then restarted on a different node. However, this is undesired behavior since it will result in a disruption to the virtual machine's guest operating system and applications. In this architecture, the cluster is configured with swap enabled and the wasp agent to manage what happens to virtual machines when the node comes under memory pressure. As a result, no soft eviction threshold is configured.

cpuManagerPolicy and topologyManagerPolicy

For virtualization workloads that benefit from predictable CPU scheduling, the kubelet's cpuManagerPolicy is set to static, ensuring that vCPUs assigned to latency-sensitive or pinned virtual machines map consistently to dedicated host CPUs. Complementing this, the topologyManagerPolicy is set to best-effort, which encourages NUMA-aware placement and improves cache locality when possible, while still allowing the scheduler flexibility to place high-density workloads even when ideal topology alignment is not achievable. This combination provides deterministic CPU allocation where required without constraining overall cluster utilization.

failSwapOn

OpenShift disallows swap by default and will prevent the kubelet from starting if swap is detected. While this behavior is appropriate for containerized workloads, it conflicts with expected hypervisor behavior and this architecture's requirement to safely support a 2:1 memory overcommit ratio for virtual machine workloads. Because VMs can rely on hypervisor-level memory accounting and recover gracefully through live migration when

pressure occurs, controlled swap usage is essential to preventing premature VM termination during short periods of host exhaustion.

By default kubelet will fail to start if swap is enabled on a host, therefore we must disable this behavior by setting the value `failSwapOn` to `false`.

Example kubelet configuration

The following is an example of `KubeletConfig` that sets all of the above values for the nodes:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: virt-kubelet-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: virt-kubelet-config
  autoSizingReserved: true
  kubeletConfig:
    kubeAPIBurst: 200
    kubeAPIQPS: 100
    maxPods: 500
    nodeStatusMaxImages: -1
    cpuManagerPolicy: static
    topologyManagerPolicy: best-effort
    failSwapOn: false
```

To apply the kubelet configuration to all nodes in the cluster, apply the label `custom-kubelet` with the value `virt-kubelet-config` to both the `master` and `worker` machine config pools. Once applied, the nodes will reboot.

Configure additional node networking

The NMstate Operator is used to configure additional network settings on cluster nodes. NMstate is a declarative interface for the Linux NetworkManager with the ability to manage all aspects of node networking. However, because RHCOS is considered an appliance, OpenShift does not support changing the configuration of the interface used by OVN-Kubernetes after node installation. If, for any reason, the IP address or other configuration of bond0 needs to be changed after the node is joined to the cluster, the node will need to be removed from the

cluster and redeployed. However, the VLAN interface used for iSCSI can be changed post-deployment.

Install the NMstate Operator

NMstate is deployed like other Operators. Follow the documentation for how to install the Operator and create an instance of the `NMState` CRD. Once instantiated, continue to the next section for configuring node networking.

Configure iSCSI network

The iSCSI network is a VLAN interface on bond0 which is used to connect to the enterprise storage array. If the iSCSI network was not configured at install time, use NMstate to configure it at this time. In addition to the NMstate snippet used in the install section of this document providing an example, an example VLAN interface configuration can be found in the documentation, ensure that appropriate IP configuration is included and, if necessary, provide route configuration so that the node understands when to use the interface.

For example, if the iSCSI network is an unrouted "flat" L2 network, meaning all endpoints are in the same broadcast domain, then additional configuration is most likely unnecessary. However, if the iSCSI network is routed and the enterprise storage device is on a different subnet, then it will be necessary to provide route configuration to the host. An example is provided below, adjust for your network topology.

```
routes:
  Config:
  # add a route for the iSCSI subnet used by the enterprise storage
  - destination: 10.4.57.0/24
    next-hop-interface: bond0.456
    next-hop-address: 10.4.56.1
```

Other host networks

Configure host-level networking using NMState by defining network bonds for virtual machine traffic and management interfaces. Establish VLAN interfaces as needed and assign them to the appropriate bonded interfaces. Ensure static routes are applied where necessary to support multi-network connectivity or traffic segregation. Include MTU settings and link aggregation policies to optimize performance and ensure consistent, fault-tolerant connectivity.

High availability

The requirements for this architecture specify that virtual machine workloads must recover from a node failure within 120 seconds or less. In OpenShift, achieving this level of availability relies on the coordinated use of node health checks (NHC) and an explicit remediation mechanism, such as Self Node Remediation (SNR) or Fence Agents Remediation (FAR), to detect unhealthy nodes and take decisive action. This section describes how these components are configured and used to ensure timely failure detection, prevent split-brain conditions, and enable rapid workload recovery consistent with the availability objectives defined earlier in this architecture.Additional details on the principles of operation and the available options for high availability are available in this KCS.

For the purposes of this architecture, the following assumptions have been made:

- Each node is equipped with a Baseboard Management Controller (BMC), or a vendor-specific equivalent, capable of independently power-cycling the host regardless of the operating system state.
- The BMC supports IPMI or an equivalent fencing interface that is compatible with Fence Agents Remediation and supported by OpenShift, enabling deterministic out-of-band power management operations.
- BMC network connectivity is reliable, routable, and reachable from the OpenShift cluster, such that remediation Pods can communicate with the BMC endpoints at all times. This may be achieved via the node management interface or an additional dedicated network (for example, an additional VLAN interface) exposed to RHCOS, however the specific host-level network configuration required to enable this access is considered out of scope for this document.
- BMC credentials and access methods are consistent and centrally manageable, allowing remediation policies to be applied uniformly across all nodes without per-host customization.

Fence Agents Remediation Operator

The Fence Agents Remediation Operator is used to communicate with the node BMC and trigger remediation action, specifically it will power cycle the node to ensure that no workload is running in order to avoid attempting to create two instances of a running virtual machine, which will lead to data corruption. To install the operator, follow the documentation. Once deployed, create an instance of the FenceAgentsRemediationTemplate to configure the agent appropriately. The example below uses IPMI, refer to the documentation for specific fence agents supported and their respective documentation for configuration parameters.

```
apiVersion: fence-agents-remediation.medik8s.io/v1alpha1
kind: FenceAgentsRemediationTemplate
metadata:
  name: far-template-fence-ipmilan
  namespace: openshift-workload-availability
spec:
  template:
    spec:
      agent: fence_ipmilan
      nodeparameters:
       # the values for the node name and IP will need
       # to be adjusted based on real values used in
       # the deployment environment
       '--ip':
          node1: 10.7.89.11
          node2: 10.7.89.12
          node3: 10.7.89.13
          # < repeat for each node >
      sharedparameters:
        '--action': reboot
        '--password': password
        '--username': admin
      retrycount: '5'
      retryinterval: '5s'
      sharedSecretName: 'fence-agents-credentials-shared'
      timeout: '30s'
```

Importantly, this is an example that is not tailored for specific vendor hardware and BMC implementations. Refer to the documentation for your vendor to determine the correct parameters to use with the remediation template.

Additionally, the documentation contains examples of how to use Secrets for per-node and shared credentials. Default OpenShift Secrets provide an obfuscated way to control the credentials, however it is likely desirable to use a secure tool for Secret management, such as the External Secrets Operator or HashiCorp Vault. Deploying, configuring, and using a Secret management tool is outside the scope of this document.

Node Health Check Operator

In this architecture, the Node Health Check (NHC) Operator is installed and configured to provide consistent, automated detection of node-level failures that directly impact virtual machine availability. Deploy the operator following the documentation, ensuring that you select

the option to enable the console plugin when deploying the Operator from the OpenShift web interface.

Once the Operator has deployed, create a node health check using either the UI or CLI. An example YAML is provided below, refer to the documentation for using the UI to configure the health check.

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nhc-far
spec:
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/worker
        operator: Exists
        values: []
      - key: node-role.kubernetes.io/control-plane
        operator: Exists
        values: []
  unhealthyConditions:
  - duration: 15s
    status: 'False'
    type: Ready
  - duration: 15s
    status: 'Unknown'
    type: Ready
  minHealthy: 70%
  remediationTemplate:
    apiVersion: fence-agents-remediation.medik8s.io/v1alpha1
    kind: FenceAgentsRemediationTemplate
    name: far-template-fence-ipmilan
    namespace: openshift-workload-availability
```

The example health check above will use the fence agents remediation template created in the previous section to power cycle nodes after the node is in the unhealthy or unknown status for 15 seconds. The control plane sets this status after five missed heartbeats from a node, which takes 50 seconds. The result is that node fencing will take place after 65 seconds with this configuration. The value of 70% for minHealthy indicates that if 30% or more of the nodes become unhealthy, the health check will not take any action. This cluster has 14 nodes, 30% unhealthy means that up to four nodes can fail before the node health check will wait for a human to intervene. This is a protection mechanism in the event of widespread failure, such as if the datacenter network is failing across multiple racks.

Cluster storage using CSI

This architecture does not provide specific guidance for which storage vendor to use nor how to configure their CSI driver to provide required connectivity, performance, and features. Using the appropriate documentation, deploy and configure the CSI driver and configure at least one storage class that is capable of meeting the needs of the virtual machines and the applications deployed within.

Importantly, the cluster must have a default storage class. This default is used whenever a workload or virtual machine does not explicitly request a specific StorageClass, including common virtualization workflows such as VM disk provisioning and when importing VM templates using import jobs. If no default storage class is configured, PVCs without an explicit storage class association will fail or be indefinitely pending, preventing virtual machines from being created or started. Follow the documentation to set the default storage class using the UI or CLI.

As laid out in the enterprise storage assumptions section above, the storage is also expected to provide snapshot and clone capabilities. Depending on your storage vendor's configuration process, the administrator may need to configure additional volume snapshot classes or create/adjust the storage profile to meet expectations and storage array capabilities.

Additional configuration

Machine config pool parallelism

Increase the machine config pool's maxUnavailable to 2 to decrease the amount of time it takes to apply updates and other configuration changes, but still stay within the acceptable failure capacity.

If multiple machine config pools (MCP) are used at any time it is important to know that each machine config pool will operate independently and simultaneously. This means, for example, that a cluster with two MCPs each with maxUnavailable set to 2 may have up to 4 nodes being updated simultaneously.

Time synchronization

Ensuring node time is synchronized across the cluster is important for a multitude of reasons. Configuring the chrony service across nodes in the cluster is done using the machine

configuration, which can be created manually or generated using Butane. Follow the documentation to configure the NTP servers to be used for the environment.

Node Maintenance and Lifecycle Configuration

The Node Maintenance Operator provides a controlled mechanism for placing nodes into maintenance mode, enabling administrators to safely drain workloads and virtual machines prior to planned infrastructure changes. In a virtualization environment, the operator coordinates node cordoning and draining in a way that minimizes disruption by allowing virtual machines to be cleanly migrated rather than abruptly terminated.

Use the documentation to deploy the Node Maintenance Operator using either the UI or CLI.

Logging

While not explicitly defined in the requirements of this architecture, it is generally recommended to use a centralized log collection and analysis tool with any OpenShift deployment. At a minimum, deploying and configuring log forwarding to an enterprise log aggregation service should be done.

Deploying the OpenShift logging service to collect and analyze logs is outside the scope of this document, however if there is sufficient compute and storage capacity available in the cluster, follow the documentation to deploy and configure this service.

Observability

Observability and logging are important capabilities for operating an OpenShift Virtualization environment, but deployment and configuration of these day 2 (operations) services is out of scope for this document. At a minimum, using the default cluster monitoring and metrics provide baseline visibility into node, storage, network, and virtualization health. Additional observability capabilities, such as those provided by the Cluster Observability Operator and Network Observability Operator, may be integrated as desired. Refer to the appropriate product documentation for specific implementation and configuration guidance.

OpenShift Virtualization

Previous sections of this document focused on the foundational infrastructure and cluster-level configuration for the architecture, this section will transition into the virtualization-specific capabilities that enable OpenShift to function as a full-featured enterprise virtualization platform. With the underlying hardware, networking, storage, and core OpenShift services configured, this will configure OpenShift Virtualization in a way that aligns with the architectural requirements of high VM density, predictable performance, and operational resilience.

Virtualization Operator

The OpenShift Virtualization Operator is the primary control plane component responsible for enabling and managing virtualization capabilities within the cluster. Its configuration is central to the behavior, scalability, and stability of virtual machine workloads. Because this architecture targets high VM density, controlled overcommit, and predictable migration behavior, the configuration is tuned to align with the design assumptions established earlier in this document. The settings described in this section focus on how the virtualization platform is shaped at a global level, ensuring that downstream VM scheduling, migration, and resource management operate consistently and at scale.

Operator deployment

Follow the documentation to install the OpenShift Virtualization Operator to the cluster. No special considerations or configuration needs to be done when deploying the Operator.

CRD configuration

The following describes the configuration to be used when creating the OpenShift Virtualization deployment, i.e. the `kubevirt-hyperconverged` CRD instance. Settings not described here are left at their default values.

Common boot image import

This cluster is internet connected and, barring any organizational security policies, using the Red Hat provided default boot images provides a quick way to begin deploying Red Hat Enterprise Linux, CentOS, and Fedora virtual machines. When `enableCommonBootImageImport` is enabled, the default storage class, configured earlier, will be used to store these boot images in the `openshift-virtualization-os-images` namespace.

If, for any reason, these images are not desired then do not enable the setting. If no default storage class has been defined, the images will not download until that configuration is applied.

Higher workload density
Ignore this setting at initial CRD configuration, it will be adjusted after configuring swap and workload density in the next section.

Default CPU model
The default CPU model will depend on the hardware in the cluster. It is used to set the CPU model used by all virtual machines, unless specifically configured in the VM definition. This value should be set to the highest common CPU type across all hardware. Since a requirement for this architecture is to have homogenous hardware this should not result in newer generation hardware having CPU features unused as a result of earlier generation hardware.

Use the qemu documentation to identify valid values for the default CPU, such as `SierraForest` or `EPYC`. If the specific CPU family is unknown, *after* launching a VM, connect to the associated Pod terminal and use the command `virsh capabilities` and look in the `capabilities.host.cpu` section to see the CPU model.

Live migration configuration
Tuning live migration is important to the architectural requirements defined for this implementation. Specifically, the shared network adapter bond0 is used for management/SDN, live migration, and iSCSI storage traffic with the goal to prevent live migration traffic from saturating the links and preventing the other services from receiving the throughput they need. As a result, the live migration traffic will be controlled by limiting the bandwidth used for live migration and the number of concurrent migrations that happen for the host.

The following configuration is used for live migration:

- `allowPostCopy` is enabled.
- `bandwidthPerMigration` = 2048Mi, limiting each migration to 2GiB of throughput, or approximately 70% of total bandwidth on a single 25Gb link.
- `completionTimeoutPerGiB` = 60, determines the amount of time elapsed before the post-copy migration is used for a VM as a result of pre-copy being unable to complete. Calculated by multiplying the total VM memory by this value.
- `network.` This does not need to be changed from the default of using the SDN.
- `parallelMigrationsPerCluster` = 5, this is the default number and represents the cluster-wide limit on simultaneous live migrations. Since there is no direct limit on the

number of in-bound migrations to a node, which could also saturate the link(s), this represents a safe starting point.
- `parallelOutboundMigrationsPerNode` = 2, when used in combination with the `bandwidthPerMigration` configuration, this effectively limits the throughput used for outbound migrations from the node.

The value for `bandwidthPerMigration` is high enough to meet the needs of all but the most demanding workloads using the pre-copy mechanism without affecting other traffic types sharing the links. However, in the event of a very busy VM that cannot complete a pre-copy migration with the available bandwidth, the `completionTimeoutPerGiB` is a failback configuration that will automatically trigger a post-copy migration after a period of time. For a VM with 16GiB of memory, this will be 16 minutes (16 GiB * 60 seconds).

Each VM is expected to have 16GiB of memory, per the requirements, and a value of 2048Mi (2GiB) for `bandwidthPerMigration` means that it is expected for a VM to take approximately 10 seconds for the memory pre-copy phase of live migration. During normal operations, each node is expected to host 83 VMs and, with `parallelOutboundMigrationsPerNode` set to 2, this means it will take approximately 420 seconds (`(83 / 2) * 10 = ~420` or 7 minutes) for data transfer to fully evacuate a node for maintenance purposes. Note that this *does not* represent the full amount of time to live migrate all VMs from the node as that depends on additional factors, including VM scheduling, Pod instantiation time (which includes the amount of time needed to attach storage), and more. However, it does estimate the floor for the minimum amount of time needed to fully evacuate a node.

Tuning policy

The moderate VM density per-node of this architecture, 83 VMs per host, might result in a high number of API operations during certain scenarios, such as recovering from multi-node failure or if the users/administrators choose to deploy many (100s) of VMs quickly. As a result, the `highBurst` profile is used to accommodate a large number of API calls in a short period of time.

CPU allocation ratio

This setting controls the CPU overcommitment ratio by determining how much physical CPU is reserved for each virtual CPU assigned to a virtual machine. The default is 10:1, which means that .1 CPU (100 millicores) will be reserved for each 1 vCPU assigned to a VM. Despite the target CPU overcommit ratio for this architecture being 4:1, this setting will not be modified. There are several reasons for this, including:

- Setting this value is tantamount to a hard enforcement of the overcommit ratio, which eliminates flexibility should unforeseen circumstances occur, e.g. temporary three+ node failure pushing the desired overcommit limits beyond the ideal threshold.
- Can reduce flexibility in workload placement, specifically when scheduling VMs, as a result of the increased CPU request. This can also lead to suboptimal results for load-aware rebalancing when the cluster is running at high utilization.
- It does not account for CPU requests associated with non-VM workloads, such as the control plane and other supporting software deployed to the OpenShift cluster.

The result is that the overcommit ratio is soft enforced by the administrator(s) and appropriate capacity planning behavior when introducing new workloads to the platform. However, if additional CPU request/reservation is needed on a per-workload basis, this should be applied using an appropriate instance type configuration..

Virtual machine compute configuration

Workload density

The requirements for this architecture include a 2:1 memory overcommit ratio, which allows nodes to host more virtual machines than their physical memory alone would otherwise permit. Without additional configuration, when a node comes under memory pressure it will begin to terminate running VMs, which is not desired behavior. To safely enable memory overcommit, we must use swap to allow the node to temporarily use more logical memory than actual memory. OpenShift disables swap by default because container workloads rely on resource requests as strict reservations - if the node begins swapping, the scheduler has no way to account for the mismatch between allocated and physically available memory. Virtual machines, however, are not subject to these constraints. Their memory is managed by the hypervisor, and with OpenShift Virtualization's ability to live migrate VMs, plus wasp agent eviction and migration mechanisms, VMs can safely operate in environments that make controlled use of swap.

To safely support overcommit, swap must be enabled and sized appropriately on each node using MachineConfig. Temporary swapping prevents premature out-of-memory events and gives the platform time to react, using wasp, when nodes approach capacity. The wasp agent continuously monitors node swap usage and migrates VMs away from overloaded nodes before swap usage becomes excessive, attempting to avoid unnecessary performance degradation. The cluster-wide setting `memoryOvercommitPercentage` configures OpenShift Virtualization to provision VMs using the desired overcommit ratio.

This architecture has already configured the kubelet with the required setting for `failSwapOn`. Enabling memory overcommit requires two additional steps:

1. Configure swap space on nodes using machine config. The nodes described earlier in this document have a single 2TB local storage device to be used for all node storage. As a result, configure a 768GiB swap file at `/var/tmp/swapfile.` This size is inline with the size determined using the calculation in the documentation.

   The `MachineConfig` used is the same as what is in the documentation, with the exception of setting the `SWAP_SIZE_MB` (line 21) value to `786432`, or 768GiB

   Note: If the physical servers have additional local disks, modify the configuration to use an appropriate disk, either wholly or with a swapfile.

2. Set the overcommit ratio in the virtualization configuration. Following the documentation, set the value of `spec.higherWorkloadDensity.memoryOvercommitPercentage` to 200. If using the GUI to enable the feature, first toggle it on, then browse to the `kubevirt-hyperconverged` CRD instance and update the value from the default 150 to the desired 200.

   The Operator will automatically deploy the wasp agent to all nodes in the cluster, no further configuration is required.

With swap configured and enabled, plus the wasp agent deployed to the nodes, safe memory overcommit is available and will result in virtual machines swapping if the node comes under memory contention before being live migrated to another node.

Load-aware workload balancing

When used with the `KubeVirtRelieveAndMigrate` profile, the descheduler enables load-aware rebalancing of virtual machines by continuously evaluating node CPU utilization and pressure to identify nodes that are overutilized. When a node exceeds the desired deviation thresholds, the descheduler triggers live migration of eligible VMs. Enabling and using load-aware rebalancing for virtual machines requires three steps:

1. Install the descheduler following the documentation.
2. Enable PSI metrics on the nodes using machine config.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-openshift-machineconfig-worker-psi-karg
spec:
  kernelArguments:
    - psi=1
```

Applying this configuration will result in the nodes rebooting.

3. Configure the descheduler using the `AsymmetricMedium` deviation threshold by default, which allows for a node to have a moderate amount of deviation from the cluster average utilization before triggering workload rebalancing.

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
 name: cluster
 namespace: openshift-kube-descheduler-operator
spec:
 managementState: Managed
 deschedulingIntervalSeconds: 300
 mode: Automatic
 profiles:
   - KubeVirtRelieveAndMigrate
 profileCustomizations:
   devEnableSoftTainter: true
   devDeviationThresholds: AsymmetricMedium
   devActualUtilizationProfile: PrometheusCPUCombined
```

The descheduler will now automatically take action to proactively migrate workload away from nodes classified as overutilized. With the `AsymmetricMedium` profile, a node with utilization 20% or more above cluster average is considered overutilized. `AsymmetricLow` and `AsymmetricHigh` change that threshold to 10% and 30%, respectively. Use the option that most closely matches the needs and expectation for how quickly the descheduler will take action in the event of an imbalance to node utilization.

Virtual machine storage configuration

Default storage classes

An OpenShift cluster can have only a single cluster-wide default storage class, however OpenShift Virtualization can configure additional storage classes for other uses. In this architecture the default storage class was defined above during configuration of the CSI driver for external storage. This storage class is used for any persistent volume claim (PVC) or VM disk which does not have a storage class specifically requested. In the absence of other configuration, it is also used by OpenShift Virtualization for scratch space, such as when doing data transfer operations like VM disk imports, and VM state storage, such as for vTPM devices.

This architecture is configured with only a single storage class that is the default for all storage types. However, if additional storage classes are added, for example to provide different performance characteristics or to consume additional enterprise storage systems, then it may be desirable to set a default storage class specifically for OpenShift Virtualization. Follow the documentation to configure a default storage class for virtualization and note that this will override the cluster-wide default storage class for all virtualization-related PVCs.

While not configured in this architecture, the OpenShift Virtualization hyperconverged CRD instance configuration allows specifying different storage classes for different data types, e.g. scratch space and state storage. If you do not wish to use the default storage class for these data types, configure appropriate values.

Storage live migration

To move VM disks between storage classes OpenShift 4.20 uses the Migration Toolkit for Containers (MTC) to orchestrate the data copy from source PVC to destination. Follow the documentation to deploy and configure the Migration Toolkit for Containers. Once configured, storage live migration can be controlled from either the MTC console or the OpenShift console.

Virtual machine networking configuration

Host network configuration for virtual machine networks

The requirements for this architecture dictate that physically separate NICs be used for virtual machine traffic. In the hardware and network architecture section of this document the decision was to use two NICs in a link aggregation bond that is dedicated to virtual machines. Since all

nodes are homogenous, the host network configuration is simplified using a single NMstate configuration that applies to all nodes.

In this architecture, physical interfaces are bonded using LACP (802.3ad) rather than balance-slb to provide deterministic link utilization, fast failure detection, and explicit coordination with the upstream switching infrastructure. While balance-slb can distribute traffic without switch configuration, it relies on source-based hashing and does not provide the same level of visibility, control, or failure semantics expected in an enterprise virtualization environment.

An Open vSwitch (OVS) bridge is layered on top of the bonded interfaces to provide the virtual switch connectivity required for virtual machines. OVS is used because it is the foundation for User Defined Networks (UDN). This approach supports common virtualization requirements such as VLAN tagging, traffic isolation, and enforcement of network policies, while remaining consistent with OpenShift's networking and security model.

A snippet of an example NMstate configuration file is provided below, however since this document does not provide exact implementation guidance it may need to be adjusted for the specific environment.

```
desiredState:
  interfaces:
  # configure MTU for NICs
  - name: enp0s1
    type: ethernet
    state: up
    mtu: 9000
  - name: enp1s1
    type: ethernet
    state: up
    mtu: 9000
  # create the bond
  - name: bond1
    type: bond
    state: up
    mtu: 9000
    link-aggregation:
      mode: 802.3ad
      port:
        - enp0s0
        - enp1s0
    ipv4:
      enabled: false
```

```
      ipv6:
        enabled: false
      lldp:
        enabled: true
    # create the OVS, use bond1 as the uplink
    - name: br1
      type: ovs-bridge
      state: up
      bridge:
        allow-extra-patch-ports: true
        options:
          stp: false
          mcast-snooping-enable: true
        port:
        - name: bond1
    # create a bridge map for UDNs to use
    ovn:
      bridge-mappings:
      - localnet: vm-network
        bridge: br1
        state: present
```

Linux bridge is not used in this architecture because it provides a more limited feature set and is not compatible with User Defined Networks. While suitable for some connectivity scenarios, Linux bridge lacks the flexibility, extensibility, and policy integration needed to support advanced virtualization use cases such as multi-network attachment, VLAN-aware segmentation, and consistent enforcement of network policies. However, Linux bridge is useful in scenarios where virtual machines require trunked network interfaces, such as virtual network and firewall appliances that need to manage VLAN tags directly.

Virtual machine networks

Cluster User Defined Networks (CUDNs) are used in this architecture to connect virtual machines to external datacenter networks in a controlled and repeatable manner. CUDNs are defined once at the cluster level and selectively applied to namespaces using label-based selectors to ensure consistent network availability. When a CUDN is made available to a namespace, the corresponding NetworkAttachmentDefinition is automatically created, allowing virtual machines to attach to the network without additional namespace or network configuration.

Creating the VM networks is done in two steps:

1. Assign appropriate labels to namespaces which have, or will have, VMs that need access to the network. This can be done before or after the CUDN is created.

   There are no rules for the label name or value, however it is suggested to use something that makes identification and management reasonable. For example, if it is expected that all namespaces with a single label have the same networks, then group them together logically. Conversely, if granular control of each CUDN is needed, then using individual labels for each one may be more appropriate.

   Follow the documentation to create Projects as needed and add labels using either the CLI or UI. Below are some example labels that can be used for CUDNs.

Use case	Example

Network groups, where each Project/namespace with the label will receive the same CUDN configuration	`oc label namespace <name> network-group=production`
	
	`oc label namespace <name> network-group=site-a`
	
	`oc label namespace <name> network-group=team-1`

Granular mapping of each VLAN to each namespace	`oc label namespace <name> vlan12=true`
	
	`oc label namespace <name> vlan34=true`
	
	`oc label namespace <name> vlan56=true`

2. Create CUDN(s) for each VLAN used by virtual machines. Use the documentation to create CUDNs using either the CLI or UI. An example using the CLI is provided below. Note that it is important to have several aspects of the configuration defined in the CUDN:
   - The namespace selector must match the labels defined in the previous step (or expected to be used if not created yet). If the selector is omitted or empty, the CUDN will be available to all namespaces.
   - The topology must be `Localnet` to have the VMs connect to the OVS and the external network. The `Layer2` topology will create a cluster-wide overlay network

that can provide L2 adjacency for Pods and VMs that are connected, however it does not enable external access using VLAN tagging.

- The physical network name must match the localnet bridge map name used when configuring the OVS in the previous section.
- IP address management (IPAM) is provided via the external enterprise services. OpenShift does not provide IPAM for these networks.
- Set the MTU to the appropriate value, ensuring that if jumbo frames are used, then all of the underlying OVS, bond, and external network infrastructure are appropriately configured.

```
namespaceSelector:
  matchLabels:
    “network-group” : “production”
network:
  topology: Localnet
  localnet:
    role: Secondary
    physicalNetworkName: vm-network
    ipam:
      mode: Disabled
    mtu: 1500
    vlan:
      mode: Access
      access:
        id: 100
```

Summary

This document presents an example architecture for running approximately 1,000 virtual machines using Red Hat OpenShift Virtualization, illustrating how traditional VM workloads can be delivered on a Kubernetes-based platform while maintaining enterprise expectations for performance, availability, and operational discipline. Rather than prescribing a single "correct" implementation, the architecture demonstrates how a set of explicit requirements, constraints, and assumptions translate into concrete design decisions across compute, storage, networking, and cluster configuration. The result is a cohesive reference that shows not only what to configure, but why those choices were made.

Adapting This Architecture

This example architecture is intentionally opinionated, but it is not static. Organizations are expected to adapt it to their own workload characteristics, infrastructure capabilities, and operational policies. When doing so, changes should be made deliberately and with an understanding of how individual design decisions influence behavior across the platform. The following areas represent the primary architectural levers that should be evaluated when adapting this design.

Virtual machine profiles and density

Changes to the average VM size (vCPU or memory) directly affect compute sizing, per-node density, and failure tolerance calculations. Larger VM profiles reduce placement flexibility and may require lower overcommit ratios, increased node counts, or adjustments to live migration concurrency and bandwidth limits. Conversely, smaller VM profiles may allow higher density but increase pressure on kubelet scaling parameters such as maxPods and API throughput.

Overcommit ratios and memory management

CPU and memory overcommit ratios are foundational assumptions in this architecture. Increasing overcommit improves utilization but raises the likelihood of contention events, which in turn places greater importance on swap performance, wasp behavior, and available headroom for live migration. Reducing overcommit simplifies operational behavior but increases hardware requirements. Any change to these ratios should be reflected holistically in swap sizing, high-availability capacity, and workload density expectations.

Storage platform and protocol selection

This architecture assumes external, CSI-provisioned storage accessed over IP-based networking. Adopting a different storage protocol or performance tier may require changes to NIC layout, bandwidth allocation, live migration tuning, and storage class design. While the core architecture remains valid, storage performance and availability characteristics directly influence VM startup time, migration behavior, and overall workload stability.

Network topology and traffic separation

Network design choices, such as link speed, VLAN layout, or the degree of physical separation between traffic types, have cascading effects on live migration reliability, storage performance, and VM network throughput. When adapting the network architecture, it is important to preserve the underlying principles of fault tolerance, predictable bandwidth, and isolation between workload and infrastructure traffic, even if the specific implementation details differ.

Control plane and node role decisions

This design uses a schedulable control plane and homogeneous nodes to maximize utilization and simplify capacity planning. Organizations that prefer dedicated control plane or infrastructure nodes should account for the resulting reduction in available compute capacity and ensure that resource reservations, failover calculations, and cluster sizing are updated accordingly.

By treating these areas as architectural inputs rather than isolated configuration changes, organizations can adapt this example implementation to a wide range of environments while preserving the core design goals of scalability, resiliency, and operational predictability.

Additional resources

In addition to finding the latest version of this guide at https://red.ht/virtrefguide, additional information regarding OpenShift and OpenShift Virtualization can be found in the documents linked below:

- OpenShift Virtualization Architecture Guide - https://red.ht/virtarchguide
- OpenShift Virtualization Cluster Sizing Guide - https://red.ht/virtsizing
- OpenShift Virtualization Hardening Guide - https://red.ht/virthardeningguide
- OpenShift Virtualization Tuning and Scaling Guide - https://access.redhat.com/articles/6994974
- OpenShift Virtualization Fencing and VM High Availability Guide - https://access.redhat.com/articles/7057929
- OpenShift Virtualization Disaster Recovery Guide - https://access.redhat.com/articles/7041594
- OpenShift Virtualization Learning Hub
- Migration Toolkit for Virtualization Learning Path
- Guide to OpenShift Virtualization as a VMware admin

Change log

December 2025, 1.0.0

- Initial release

Appendix

Table: Key Performance and Capacity Metrics for OpenShift Virtualization		
Category	**Metric Name(s)**	**Purpose / Description**
Compute Metrics	`node_cpu_seconds_total`	Tracks host CPU load and identifies CPU saturation or imbalance across worker nodes.
	`kubevirt_vmi_vcpu_seconds_total`	Measures per-VM CPU usage to detect overcommitment or pinned-core contention.
	`container_cpu_cfs_throttled_seconds_total`	Detects contention caused by CPU quotas or insufficient CPU isolation.
	NUMA alignment and CPU pinning compliance	Ensures VMs with dedicated CPUs are correctly aligned with host topology to minimize latency.
Memory Metrics	`node_memory_MemAvailable_bytes`	Detects memory overcommitment and resource pressure on host nodes.
	`kubevirt_vmi_memory_used_bytes`	Monitors guest VM memory allocation and identifies excessive consumption or memory leaks.
	`node_memory_HugePages_*`	Verifies hugepage reservation and usage for latency-sensitive or high-performance workloads.
	`node_vmstat_pswpin`, `node_vmstat_pswpout`	Indicates swap activity, signaling potential host memory exhaustion or misconfiguration.
Storage Metrics	`kube_persistentvolumeclaim_*`, `kubevirt_vmi_storage_*`	Tracks VM disk IOPS and throughput to detect performance bottlenecks.
	`volume_*` (CSI driver metrics)	Monitors backend SAN or storage driver latency and throughput for performance validation.

	`etcd_disk_wal_fsync_duration_seconds`	Measures etcd write latency — critical for overall cluster control-plane responsiveness.
Network Metrics	`node_network_receive_bytes_total`, `node_network_transmit_bytes_total`	Monitors overall NIC and bond interface throughput on each host.
	`kubevirt_vmi_network_receive_bytes_total`, `kubevirt_vmi_network_transmit_bytes_total`	Identifies VM network utilization trends and potential congestion.
	`node_network_dropped_total`, `node_network_errors_total`	Detects network misconfiguration, VLAN mismatch, or MTU inconsistency across interfaces.
Cluster & Virtualization Health	`kubevirt_vmi_phase_count`	Tracks the lifecycle state of VMs (running, paused, failed) for operational visibility.
	`kubevirt_vmi_migration_duration_seconds`	Measures live migration performance, duration, and success rates.
	`kube_node_status_condition`	Monitors node health and readiness to detect infrastructure issues affecting VM availability.
	`kube_apiserver_request_duration_seconds`, `etcd_request_duration_seconds`	Monitors control-plane responsiveness, ensuring reliable API and etcd performance.