OpenShift Virtualization

Cluster Sizing Guide

Hybrid Platforms Business Unit

version: 3.1.0

June 2026

Legal Notice

Red Hat

Table of Contents

Red Hat

About this Document

Purpose of this document

This document provides a step-by-step example for sizing an OpenShift cluster to host virtual machine workloads using OpenShift Virtualization. It covers determining raw resource requirements, applying overcommitment ratios, selecting node sizes, choosing a control plane topology, and establishing a node count that satisfies high availability requirements.

The values used throughout are illustrative. Where real workload data exists, for example from an existing virtualization platform, it should be used in place of the example VM sizes. Where it does not, the example sizes serve as reasonable starting points.

Sizing is not a one-time exercise. Once the cluster is operational, the monitoring and capacity management tools available in OpenShift should be used to validate the assumptions made here and to adjust capacity as utilization patterns become clear.

Prerequisites and target audience

This document is intended for infrastructure architects and platform engineers responsible for planning an OpenShift Virtualization deployment. Readers should be familiar with the following:

- Core OpenShift Container Platform concepts: nodes, pods, and the kubelet.
- General virtualization concepts: vCPU allocation, memory overcommitment, and live migration.
- The organization's workload profile and acceptable risk tolerance for resource contention.

No prior experience with OpenShift Virtualization-specific sizing is required. The document walks through each decision step by step, with the reasoning behind each choice explained alongside the calculations.

Red Hat

OpenShift Virtualization Cluster Sizing

Step 1: Determine the raw capacity required for workloads

The first step is to establish the total raw resource demand of the workload by summing each of the CPU and memory requirements that all virtual machines require, before any overcommitment or cluster overhead is considered. This is the foundation for every calculation that follows, so using VM sizes that accurately reflect the expected workload is important. If data is available from an existing virtualization environment, use real utilization figures rather than the t-shirt sizes below. In the absence of real data, the following sizes are reasonable starting points:

- Micro: 1 vCPU, 1 GiB memory
- X-small: 1 vCPU, 4 GiB memory
- Small: 2 vCPU, 8 GiB memory
- Medium: 4 vCPU, 24 GiB memory
- Large: 8 vCPU, 48 GiB memory
- X-large: 16 vCPU, 64 GiB memory
- XX-large: 32 vCPU, 96 GiB memory

For this example, the target workload is:

- 1,000 small (2 vCPU, 8 GiB)
- 300 medium (4 vCPU, 24 GiB)
- 200 large (8 vCPU, 48 GiB)
- 200 x-large (16 vCPU, 64 GiB)

CPU

The sum of vCPU requirements across all VM types results in the total raw CPU need:

VM size	Count	vCPU each	Total vCPU
Small	1000	2	2000
Medium	300	4	1200
Large	200	8	1600

X-large	200	16	3200
Total	1700		8000

Memory

Memory overhead for each VM is calculated using a documented formula, which accounts for the memory consumed by the platform and KVM hypervisor functions that run alongside each VM. Each virtual machine in this example is assumed to have a single graphics device.

```
Memory overhead per VM ≈ (0.002 × requested memory in MiB) + 218 MiB + (8 MiB × number of vCPUs) + (16 MiB × number of graphics devices)
```

This formula is then applied to each VM type to determine the overhead requirements. The total memory requirement, including overhead, is calculated to be:

VM size	Count	Memory per VM	Total VM memory	Overhead per VM	Total overhead
Small	1000	8GiB	8000GiB	266.384MiB	260GiB
Medium	300	24GiB	7200GiB	315.152MiB	92GiB
Large	200	48GiB	9600GiB	396.304MiB	77GiB
X-large	200	64GiB	12800GiB	493.072MiB	96GiB
Total	1700		37600GiB		525GiB

The total memory required for the workload is the sum of the VM requirements plus the overhead: 38125GiB.

Average virtual machine size

The following steps use the average VM size to estimate per-node capacity. To determine the average size, we divide the total by the number of VMs:

- Average vCPU: 8000 / 1700 = 4.7 vCPU
- Average memory: 38125 GiB / 1700 = 22.4 GiB

  Note: The average VM size is used for total capacity planning, however if the virtualization deployment has outlier virtual machine sizes, such as those that are

drastically larger than the average, they may be affected by this approach. For example, depending on how aggressively utilized the cluster nodes are, very large VMs may be unable to schedule without taking additional steps to assist with workload management.

Step 2: Factor in overcommitment for CPU and memory

Overcommitment allows more virtual resources to be allocated to VMs than the physical hardware can simultaneously satisfy. When all VMs are active, but not using all their resources at the same time, overcommitment improves hardware utilization without affecting performance. The trade-off is that if contention occurs, VMs will experience performance degradation. Depending on other features configured in the cluster, host memory pressure might result in VMs being live migrated to other nodes, VM memory being swapped to disk, or VMs being shutdown on the node with pressure and restarted on another node. Choosing overcommit ratios is a business risk decision, not a platform configuration decision, and the right values depend on the workload characteristics and the organization's tolerance for performance degradation under contention.

CPU overcommit

OpenShift Virtualization has supported CPU overcommit since OpenShift 4.5, with a default maximum of 10:1. The platform implements this by assigning a resource request of 100 millicores (0.1 CPU) for each vCPU allocated to a VM, meaning that 10 vCPUs can be running for each 1 physical core. CPU overcommit is appropriate for workloads that are bursty or have low average utilization, where the probability of all VMs demanding full CPU simultaneously is low. It is less appropriate for latency-sensitive or consistently CPU-bound workloads.

The `cpuAllocationRatio` setting controls the platform's CPU resource request per vCPU and defaults to 10:1. This is not the same as the overcommit ratio used for capacity planning in this document. Changing `cpuAllocationRatio` to match the planning ratio would result in a scheduling constraint, in the form of a CPU request, on every VM which has the effect of potentially reducing placement flexibility and interfering with load-aware rebalancing. The planning ratio should be treated as a soft capacity target enforced through administrator discipline, not by modifying the OpenShift Virtualization configuration.

Memory overcommit

Memory overcommit has been supported since OpenShift 4.17. To use memory overcommit, swap must be configured on each node to safely handle when VM memory demand exceeds physical RAM. The wasp agent component monitors node swap usage and live migrates VMs away from nodes approaching exhaustion before performance degrades significantly. Without

swap, memory overcommit is strongly discouraged. If physical memory is exhausted the node's OOM killer will terminate processes, potentially including running VMs.

The configuration is controlled by a single cluster-wide setting, `memoryOvercommitPercentage`. Unlike most virtualization platforms, which express overcommit as a ratio of total allocated VM memory to physical RAM, OpenShift expresses it as a percentage of each VM's assigned memory that is reserved for the VM when scheduling. When no overcommit is configured, each VM has a request (akin to a reservation) for it's full amount of memory. The overcommit percentage reduces that request proportionally. The VM memory allocation that counts towards the node's utilization when scheduling is calculated as:

```
capacity per VM = assigned memory x (100 / memoryOvercommitPercentage)
```

A setting of 200% means OpenShift counts half of each VM's configured memory against the node's available capacity when placing VMs, allowing twice as many VMs to be scheduled on a node as the physical RAM would otherwise support. The table below shows how common settings map to the ratio-based framing familiar from other platforms:

Setting	Equivalent ratio	Capacity counted per VM
100%	1:1	100% of configured memory
150%	1.5:1	67% of configured memory
200%	2:1	50% of configured memory
300%	3:1	33% of configured memory
400%	4:1	25% of configured memory

Regardless of how overcommit is expressed, as a ratio or as a percentage, allocation ratios measure configured memory against physical capacity, not actual memory used of the VMs against available node memory. The variable that determines whether memory pressure occurs is the working set: the amount of memory each VM is actively using at any given moment. A cluster with a 4:1 overcommit ratio where VMs collectively use 20% of their configured memory may be under no memory pressure at all. A cluster at a 2:1 ratio where VMs are consistently using nearly all of their configured memory will experience swap pressure regularly.

OpenShift also supports kernel same-page merging (KSM) and free page reporting to increase effective memory density further, however these are not accounted for in this sizing exercise as predicting their precise effectiveness is not possible.

Choosing overcommit ratios

Determining what overcommitment ratios to use for CPU and memory is based on the acceptable risk profile for the business. Put differently, overcommitment means that when there is contention for resources the VMs will have reduced performance as a result of CPU throttling and/or memory swapping. For some workloads, this may be acceptable, for others it may not.

When choosing an overcommit ratio:

- Use real utilization data if available. If an existing virtualization platform is hosting the workload, gather the average and peak CPU and memory utilization information for the virtual machines. This will provide the most accurate estimate of the overcommit ratios that can be safely used.
    - To estimate the memory overcommit value to use with OpenShift, look at the allocated memory of the virtual machines and divide that by the amount of memory in use on the hypervisor nodes. For example, if the sum of VM allocated memory is 8 TiB and the hypervisors have 3 TiB of memory used, `8 / 3 = 2.66:1` overcommit ratio for capacity planning purposes. To convert this to a percentage, multiply by 100, which results in 266%.
- CPU and memory contention have different consequences that may affect your decisions. CPU contention results in throttling, which is evident by increased latency. In moderation, this is tolerable for many workloads, so long as they are not latency sensitive. However, memory contention results in swapping, which will have a significantly higher impact to the applications. Worst case scenario, if overcommit is used and swap is not configured, the virtual machine(s) can be forcefully terminated, which is akin to a crash from the perspective of the guest operating system.
- Higher overcommit ratios increases the importance of headroom. Higher overcommit ratios make it more critical to maintain capacity for high availability, configure wasp thresholds correctly, and ensure the load-aware descheduler is active to prevent hot spots from developing.

In this example, we will use values of 4:1 CPU overcommit and 2:1 memory overcommit, values which represent a mixed workload with moderate-to-average utilization. To determine the adjusted physical resources required, divide the raw totals from step 1 by the respective overcommit ratios:

Resource	Raw total	Overcommit ratio	Physical capacity required
CPU	8000 vCPU	4:1	2000 cores
Memory	38125 GiB	2:1	19063 GiB

This represents the total amount of physical resources needed to accommodate the workload in the cluster. After accounting for overcommit, the adjusted average VM size is 1.2 cores and 11.2 GiB of memory each.

Step 3: Identify node size

Node size is determined by balancing four considerations:

1. The density of VMs per node and its effect on blast radius in a failure scenario.
2. The CPU and memory ratio of candidate nodes relative to the workload's requirements.
3. The impact of hyperthreading / SMT on effective CPU capacity.
4. The overhead consumed by the node itself before any VM is scheduled.

These considerations affect each other, so it is easiest to understand each one, determine the ideal outcome for the business and applications, then evaluate a range of candidate node sizes against the workload requirements and select the one that best meets the requirements.

The calculations in this step use the post-overcommit average VM size, calculated in step 2, of 1.2 physical cores and 11.2 GiB memory per VM.

Density

There are two factors which affect the density of VMs that each node hosts:

1. The per-node maximum density is determined by the `maxPods` kubelet setting, which defaults to 250 in OpenShift, but can be increased to 500 with the default SDN configuration. Each VM consumes one Pod, so this represents the maximum number of VMs which a single node can host. While technically possible, going beyond 500 VMs per node is discouraged as it can affect kubelet stability and API server load, regardless of whether physical resources are available.
2. The logical maximum is a separate, lower limit set by the organization based on acceptable risk factors, specifically the number of VMs that must be restarted on

surviving nodes if a single node fails, a.k.a. blast radius, and the impact it has on the business. A node hosting 500 VMs represents a much larger failure event than one hosting 100. The logical limit should be determined before sizing begins, with factors such as recovery time and impact to the business affecting the ideal maximum VMs per node.

CPU:memory ratio

To avoid stranding resources, where all of one resource (CPU or memory) is used while there is still available capacity for the other resource, the node's hardware should be configured so that CPU and memory are exhausted at approximately the same rate. The easiest way to do this is to determine the ratio of CPU to memory for the average VM:

```
Target ratio = average physical cores per VM : average memory per VM
             = 1.2 : 11.2
             = 1 : 9.3
```

This means that for every effective core on the node, approximately 9.3 GiB of memory is needed to keep resource usage balanced. Hardware that deviates significantly from this ratio will have one resource exhausted while the other remains partially unused. The table below provides this ratio to highlight node hardware configurations which reduce the likelihood of stranding resources.

Hyperthreading / SMT

Physical CPUs expose multiple threads per core, known as hyperthreading with Intel technology and SMT with AMD, however a single core running two threads does not provide double the compute capacity. Determining the amount of capacity to add for CPU threads is based on several factors, including:

- CPU-bound vs IO and memory bound workloads. IO- and memory-bound workloads benefit most from hyperthreading / SMT. CPU-bound workloads benefit the least.
- CPU cache pressure and working set size affect the effectiveness of hyperthreading / SMT.
- CPU-intensive workloads, particularly when two CPU-intensive VMs are running on the same core, can reduce the effectiveness of hyperthreading / SMT.
- High total CPU utilization results in the performance benefits of hyperthreading / SMT being reduced.

Real utilization data collected from running workloads is the most accurate and valid indicator of the expected capacity increase from using hyperthreading / SMT. If that data does not exist, consider these ranges based on your risk tolerance:

- Conservative - 10-15% additional capacity
- Moderate - 20-30% additional capacity
- Aggressive - 35-45% additional capacity

For this document, a common and reasonable convention is to count each additional thread as 25% additional capacity, or 0.25 cores of additional capacity per thread. As noted above, the real world value will depend on your workloads. Using this value, the formula for calculating effective CPU capacity for a node is:

```
Effective cores = physical cores + ((total threads − physical cores) × 0.25)
```

For example, a server with 64 physical cores and 128 threads has 64 + ((128 − 64) × 0.25) = 80 effective cores. This is the value we will use for node capacity when determining how many VMs can be hosted by each server.

> **Note:** There are many reasons to use, or not use, hyperthreading/SMT. Each organization should make the decision based on their security posture, performance requirements, and risk tolerance in the event of resource contention. This document does not provide guidance on the decision to use, or not use, this technology.

Node sizing and overhead

Thus far the calculation has not taken into account node overhead because the node size determines the overhead and the previous information factors into the ideal node size. Node overhead has two components: system and kube-reserved resources (managed by kubelet), and a fixed OpenShift Virtualization overhead of 2 cores and 360 MiB per compute node.

Node CPU	Effective size	Overhead	Workload capacity	CPU:mem ratio	Workload node count	VMs per node
16C / 32T	20C / 256GiB	2.5C / 12.1 GiB	17.5C / 243.9 GiB	1:13.9	122	14
32C / 64T	40C / 512GiB	2.5C / 18.9 GiB	37.5C / 493.1 GiB	1:13.2	55	31

48C / 96T	60C / 768GiB	2.8C / 24.6 GiB	57.2C / 743.4 GiB	1:13.0	37	47
72C / 144T	90C / 1024GiB	3.1C / 29.7 GiB	86.9C / 994.3 GiB	1:11.4	24	72
96C / 192T	120C / 1536GiB	3.5C / 40.4 GiB	116.5C / 1495.6 GiB	1:12.8	18	97
128C / 256T	160C / 2048GiB	4.0C / 52.2 GiB	156.0C / 1995.8 GiB	1:12.8	14	121
256C / 512T	320C / 4096GiB	5.9C / 94.7 GiB	314.1C / 4001.3 GiB	1:12.7	7	261
384C / 768T	480C / 8192GiB	7.8C / 176.1 GiB	472.2C / 8015.9 GiB	1:17.0	5	393

Calculating the workload node count is a two step process. First, the maximum VMs per node, which is not in the table above, is calculated by determining which resource, CPU or memory, would be exhausted first. To calculate the values for CPU and memory:

- Round down `workload_cpu_capacity / average_vm_cpu` to determine the maximum VMs before CPU is exhausted.
- Round down `workload_memory_capacity / average_vm_memory` to determine the maximum VMs before memory is exhausted.

Once the maximum number of VMs is determined by taking the lesser of the two values above, step two is to divide the total VM count by that number and round up. This determines the number of workload nodes that are needed.

The VMs per node value in the table above is determined by dividing the total VM count by the workload node count as described above.

Choosing a node size

The table above simplifies the process of comparing server size options for the workload. There are several important things that become easier to notice:

- The CPU:memory ratio for the workload is 1:9.3, however none of the node sizes match this exactly with the closest options being the nodes with a 1:11.4 ratio. This increases the likelihood of unused memory being present on the nodes.

- The failure domain, i.e. the number of VMs that must be restarted if a node fails, grows with node size. Put differently, larger nodes have more VMs which means if a node fails more VMs are affected. A 480C node hosts 393 VMs, meaning a single failure displaces 23% of the entire fleet simultaneously and may result in extended recovery times depending on other factors. The 160C node hosts 130 VMs per node, which may be a more manageable failure event. This is both a business decision and a technology decision, where the business determines the impact of losing so many VMs simultaneously, but the full technology stack (storage, network, application, etc.) may have complicating effects on the recovery regardless of the number of VMs affected. Larger nodes reduce cluster node count and operational complexity but increase the blast radius of any individual failure.
- While not reflected directly in the table, it's important to factor overcommitment into the node CPU:memory ratio, the impact it has to resources, and the decision for which node size to use. Higher overcommit ratios mean there is more likelihood of bursty workloads, therefore having some amount of capacity available is important. The ideal ratio described above assumes a perfect world where resource consumption falls inline with overcommit ratios. The ratios for the servers in the table all show a configuration where CPU will be the first resource exhausted, however if real world memory overcommit is less than the expected 2:1, then this can be a valuable safeguard against resource shortages. The exception is when no overcommit is being used. In this instance, closely adhering to the ratio is ideal for avoiding stranded resources.
- Finally, subscription costs may be a relevant factor for bare metal deployments. OpenShift subscriptions on bare metal are priced per node, meaning fewer, larger nodes reduce the total subscription footprint compared to a greater number of smaller nodes hosting the same workload. For organizations where this is a significant consideration, it may tip the balance toward a larger node size than would otherwise be preferred on purely technical grounds.

For this sizing example, we will choose the 160C / 2048GiB server size as a compromise between cluster node count and failure domain size. The 90C / 1024GiB size is closer to the desired ratio, however 10 additional nodes in the cluster has other considerations for (and impact to) operational complexity, power/space/cooling in the datacenter, and cost efficiency of higher density nodes.

As a result, the cluster needs 14 servers of 128 cores (160 cores effective capacity) and 2048GiB memory to host the workload. These servers will each host 121 virtual machines, totaling 145 cores and 1355GiB of memory used. This results in 93% CPU and 69% memory used for each node.

A 14 node cluster will have a total of 2240 cores of CPU and 28672 GiB of memory capacity. As determined in step 2, the workload will consume 2000 cores and 19062 GiB of cluster capacity and overhead, determined in step 3, will consume an additional 56 cores and 731 GiB of memory. This brings total cluster utilization to 92% CPU and 69% of memory.

Failure domain

Failure domain refers to the amount of resources affected by a node failure. For a cluster with 14 nodes, a single node failing represents 7% of total cluster capacity and, approximately, 121 virtual machines that will need to be restarted on the surviving nodes. A decision would need to be made as to whether this is too large (or too small) of a failure domain. If it is too large, then additional nodes would need to be added to the cluster or nodes with less capacity could be used to increase the total count.

For example, the 90C / 1024GiB node size creates a cluster of 24 nodes, each one being just 4% of cluster capacity, hosting 72 VMs each. As was mentioned above, this has an impact in other areas that results in a business decision regarding the trade offs between cost, complexity, and impact to the business.

For this example, the 14 node cluster represents an acceptable failure domain for this sizing exercise.

Step 4: Adjust for cluster architecture

OpenShift supports several control plane topologies. The right choice affects how many nodes your cluster requires and which features are available.

- Single Node OpenShift (SNO) runs the entire cluster, including control plane, infra, and workloads, on a single node. It is intended for edge deployments and is not suitable for production virtualization at scale. A SNO deployment can have additional compute nodes, however for scale and availability reasons this is not a good choice for this example.
- Two-node cluster with arbiter (GA in OCP 4.20) provides a two-node topology where both nodes are schedulable and a lightweight third node acts as an arbiter to maintain quorum. This topology is designed for resource-constrained environments where a three-node control plane is not an option. This topology does not support additional compute nodes and is, therefore, unsuitable for this example.

- A schedulable control plane runs control plane components on nodes that also host applications and virtual machines. This potentially reduces the total node count but is not supported when swap-backed memory overcommit (wasp agent) is enabled. Enabling swap on control plane nodes is an unsupported configuration.
- A dedicated control plane uses three (or four or five) nodes exclusively for control plane components, with all VM workloads scheduled on separate worker nodes. Refer to the architecture guide for more information about when to use three, four, or five control plane nodes.
- Using a hosted control plane (HCP) moves the control plane components off the cluster entirely and runs them on a separate management cluster. From the perspective of this cluster, there are no control plane nodes to size, only worker nodes hosting virtual machines and infra workloads. HCP is the recommended path when memory overcommit is enabled and operational complexity of a management cluster is acceptable.

This example uses memory overcommit, therefore the recommended architecture is to use hosted control planes. This means that additional capacity for the control plane does not need to be accounted for within the workload cluster's nodes.

If a management cluster is not available, or hosted control planes are not an option, use a dedicated control plane instead. Add three or five dedicated control plane nodes (refer to the architecture guide for details on how to choose) to the total footprint. These nodes are sized separately from the workload nodes and do not reduce the workload capacity calculated previously. Refer to the OpenShift control plane sizing documentation for control plane node sizing guidance based on your expected node count, pod density, and API load.

Infrastructure workloads

Regardless of control plane topology, infrastructure workloads, such as the ingress controllers, internal image registry, cluster monitoring stack, and any optional components such as logging, must be scheduled somewhere. Whether using a hosted control plane or dedicated control plane deployment, these features run on worker nodes unless dedicated infra nodes are provisioned.

Homogenous clusters reduce management overhead and complexity, which makes them the preferred choice. Having dedicated infra nodes that are the same size as the compute nodes would not be an effective and efficient use of capacity, therefore this example will co-host the infra workloads with the virtual machines. This has the effect of reducing the capacity available for virtual machines on the compute nodes.

Sizing the infra workloads is outside the scope of this document, refer to the architecture guide for additional information about how to choose between dedicated vs shared infra nodes. The list below contains common infra components, use the relevant documentation to help size the various infra components for creating more precise requirements.

- Ingress controllers
- Internal image registry
- Cluster monitoring and observability
- Cluster logging
- GitOps
- CSI drivers and agents
- 3rd party monitoring agents

For this example, we assume a total infra workload requirement of 24 cores and 128 GiB memory, spread across the 14 workload nodes. This adds approximately 1.7 cores and 9.1 GiB per node, which is well within the headroom available on the selected 128C/256T nodes (149.2 cores and 1,407.4 GiB consumed by VM workload and node overhead, out of 156.0 cores and 1,995.8 GiB available workload capacity). No additional worker nodes are required to accommodate infra workloads.

> **Important:** The number used for infra workload resources is not based on validated numbers and is provided here for illustrative purposes only. You must use the documentation for each of the infrastructure and control plane services to determine how much resources are needed based on your expected deployment scenario and requirements.

Step 5: Adjust for high availability and avoiding resource contention

Avoiding resource contention

Resource contention, such as nodes experiencing memory contention, can result in VMs being swapped or terminated. In this example, memory overcommit is being used, which will result in VMs being swapped, and potentially live migrated, if the node is experiencing memory pressure. This is safe and expected behavior. However, if memory overcommit is not being used it is important to consider a soft eviction threshold for memory utilization to allow VMs to be safely shutdown and restarted on a different node instead of being terminated by the OOM_kill process. Refer to the architecture guide for information about setting the soft eviction threshold and values to consider.

Red Hat

Node failure tolerance

This example considered the impact of failure domain / blast radius above, however this is not the same as failure tolerance and recovery capacity. Choosing the number of node failures to tolerate is an important consideration with regard to both high availability of the workload and the ability to complete maintenance activities. For example, performing a cluster update will result in at least one node being unavailable for workloads while updates are applied. Capacity to move / rehost workload from an unavailable node onto other nodes must be available or the cluster update cannot succeed. Refer to the architecture guide for additional considerations and decision points regarding the number of node failures to tolerate.

To determine the capacity needed for failure, start by understanding how lost capacity affects the cluster's maximum utilization during normal operations. Using an example, a 10-node cluster is hosting 100 virtual machines at 80% utilization, representing 800 of 1000 units of capacity being used. If one node fails, there are now 9 nodes remaining to host the 100 virtual machines, which still need 800 units of capacity in a cluster which now has only 900 available. Total cluster utilization has increased from 80% of 10 nodes to 89% of 9 nodes.

Continuing the example, a 10-node cluster hosting 100 virtual machines using more than 90% capacity, meaning more than 900 of 1000 units of capacity are being used, cannot tolerate node failure. A 9 node (900 capacity unit) cluster cannot restart all failed virtual machines if they total more than 900 units of capacity.

Expanding to two node failures, a 10-node cluster cannot restart VMs totaling more than 800 units of capacity post-failure. The 200 units of lost capacity represents 25% of the surviving cluster's capacity.

Returning to this sizing example, the established minimum cluster size for workload is 14 nodes, the below table shows the cluster utilization cap that applies for various cluster sizes and failure counts.

Cluster size	1 node failure utilization cap (n-1/n)	2 node failure utilization cap (n-2/n)	3 node failure utilization cap (n-3/n)
14 nodes	93%	86%	79%
15 nodes	93%	87%	80%

16 nodes	94%	88%	82%

To determine the best fit, calculate total cluster utilization and map it against the cluster sizes and failures tolerated above. The real cluster utilization must be at or below the failure utilization cap for a particular cluster size and failure tolerance scenario to be usable.

Total cluster utilization is calculated by adding together the previously defined resource requirements:

- Step 2 determined that virtual machines will use 2000 CPU cores and 19063 GiB memory.
- Step 3 determined that, with the node size selected, overhead will consume 56 cores and 731 GiB of memory.
- Step 4 identified that 24 CPU cores and 128 GiB of memory are needed for infra workloads.

Total resource requirements are 2080 cores and 19922 GiB of memory. To determine if the cluster capacity can accommodate the requirements and failure, calculate the percentage of utilization against the sum of cluster resources.

- 14 nodes have 2240 cores and 28672 GiB memory. The resource requirements would consume 93% of cluster CPU and 69% of cluster memory capacity. This exactly matches the utilization maximum for single node failure, however it exceeds the threshold for 2 or more nodes failing.
- 15 nodes have 2400 cores and 30720 GiB memory. The resource requirements would consume 87% of cluster CPU and 65% of cluster memory capacity. This is below the utilization threshold for both 1 and 2 node failures.
- 16 nodes have 2560 cores and 32768 GiB memory. The resource requirements would consume 81% of cluster CPU and 61% of cluster memory capacity. This is below the utilization threshold for 1, 2, and 3 node failures.

In all three instances, CPU is the resource that determines the utilization cap. To identify the cluster sizes needed for failure scenarios, calculate the utilization percentage for each node count (14, 15, 16 nodes) against the workload requirements and failure tolerance. Columns where the node failure utilization cap is the same or higher than the workload utilization per node represent eligible cluster sizes for the failure tolerance.

Cluster size	Workload CPU utilization / node	1 node failure utilization cap	2 node failure utilization cap	3 node failure utilization cap
14 nodes	93%	93%	86%	79%
15 nodes	87%	93%	87%	80%
16 nodes	81%	94%	88%	82%

A cluster with 14 nodes will tolerate 1 node failure, 15 nodes will tolerate 2 nodes of failure, and 16 nodes will tolerate 3 nodes of failure.

For this example, the expectation is that the cluster will tolerate 2 nodes of failure. When all 15 nodes are healthy they will each host `1700 / 15 = 113` virtual machines, with resource utilization of 136 cores (85%) and 1266GiB memory (64%) for the virtual machines.

Additional information about configuring high availability can be found in the documentation and this KCS.

Step 6: Storage, network, special resources, and more

This step is a placeholder to ensure that storage, network, and special resource requirements are not overlooked. The sizing performed in steps 1–5 addresses CPU and memory only. There are multiple other aspects to consider that may affect specific node hardware and cluster geometry. An incomplete list includes:

- CPU, memory, network, and disk requirements for Kubernetes-native storage solutions. See https://red.ht/workswithvirt for information about Kubernetes-native software-defined storage solutions that work with OpenShift.
- Network throughput for cluster functions, SDN, live migration, storage traffic, hosted applications, etc.
- High IO/throughput workloads that may cause resource starvation for co-hosted workloads.
- GPUs, SR-IOV devices, AI accelerators, and other "special" resources may be allocated to virtual machines such as dedicated CPUs, NUMA regions, etc.
- Regulatory or self-imposed scheduling requirements/restrictions.
    - Are there requirements that affect the ability to fully utilize the resources in the cluster? For example, "Team A's VMs cannot be co-hosted on the same servers as Team B's." This includes not just (anti)affinity rules but any taints/tolerations

and special resources that the customer desires to protect. These will affect overall resource utilization and may impact sizing.

- Third-party agents and services, such as monitoring and log forwarding.
- Will there be excessively large virtual machines that might be a challenge to schedule? Will there be VMs with substantial CPU requirements that may need special accommodations and/or have limited scheduling opportunities?

Each of these have their own resource requirements and considerations that need to be added to the CPU/memory requirements documented here and/or accounted for separately. The architecture guide contains guidance and suggestions for how to factor these aspects into your solution and the example architectures have tangible examples that include these aspects as well.

(Optional) Step 7: Disaster recovery

Disaster recovery planning involves additional sizing considerations beyond the scope of this guide, including secondary cluster capacity, replication bandwidth, recovery time objectives (RTO), and recovery point objectives (RPO).

OpenShift Virtualization 4.21 introduces cross-cluster live migration (GA), which enables workload mobility between clusters and may affect both primary and secondary cluster sizing.

For full disaster recovery sizing guidance, refer to the OpenShift Virtualization Disaster Recovery Guide.

Final result

Summarizing the steps and their results:

- Step 1 – Raw capacity: The workload comprises 1,700 VMs across four t-shirt sizes. Total raw vCPU demand is 8,000 vCPUs and total raw VM memory including per-VM overhead is 38,125 GiB, giving an average of 22.4 GiB per VM..
- Step 2 – Overcommit: A CPU overcommit ratio of 4:1 and a memory overcommit ratio of 2:1 were applied, reducing the physical capacity requirement to 2,000 cores and 19,063 GiB. Memory overcommit is backed by the wasp agent using node swap. The post-overcommit average VM profile is 1.2 cores and 11.2 GiB, giving a target CPU:memory ratio of 1:9.3.

- Step 3 – Node size: The 128C/256T node (160 effective cores, 2,048 GiB) was selected. After subtracting node overhead of 4.0 cores and 52.2 GiB, each node provides 156.0 cores and 1,995.8 GiB of workload capacity. The minimum workload node count is 14, with a maximum density of 130 VMs per node and an operating density of 121 VMs per node.
- Step 4 – Cluster architecture: Memory overcommit via the wasp agent is incompatible with schedulable control planes, as a result hosted control plane (HCP) was selected. Control plane capacity is accounted for on the management cluster and does not affect this cluster's node count. Infra workloads are assumed to require 24 cores and 128 GiB, distributed across the workload nodes.
- Step 5 – High availability: N+2 failure tolerance was required. At 14 nodes, normal operating CPU utilisation is 92.9%, which exceeds the N+2 cap of 85.7%. Adding one node brings the cluster to 15 nodes, reducing node utilisation to 86.8%, within the N+2 cap of 86.7%. The final workload node count is 15.
- Steps 6 and 7 – Storage, network, special resources, and disaster recovery are outside the scope of this sizing exercise and must be addressed separately.

Workload	1,000 small (2 vCPU / 8 GiB), 300 medium (4 vCPU / 24 GiB), 200 large (8 vCPU / 48 GiB), 200 x-large (16 vCPU / 64 GiB)
Total VMs	1700
CPU overcommit ratio	4:1
Mem overcommit ratio	2:1
Physical CPU required (for VMs)	2000 cores
Physical mem required (for VMs)	19,063 GiB
VM memory overhead	525 GiB
Infra workload	24 cores, 128 GiB
Node size	128C/256T (160 effective cores / 2,048 GiB)
Per-node overhead	4.0 cores / 52.2 GiB
Failure tolerance	2 node

Cluster node count	15
Nominal operating CPU utilization	87%
Nominal operating mem utilization	65%

Further considerations

The nominal CPU and memory utilization reflects the CPU:memory ratio that was chosen for the node size. On an individual node basis this does not seem like much, however at the aggregate we see that it results in significant potentially unused resources. The architect may consider a different node size or altering the overcommit ratios if they want to more closely align the two numbers, or this may be acceptable based on expected growth and utilization patterns within the organization.

Appendix: Sizing checklist

STEP 1 – Raw Capacity

- List all VM sizes (vCPU + memory) and counts
- Sum total vCPU across all VMs
- Calculate per-VM memory overhead
- Sum `total VM memory + total overhead`
- Compute average VM size: `total vCPU / VM count`, `total memory / VM count`

STEP 2 – Overcommitment

- Choose CPU overcommit ratio
- Choose memory overcommit ratio (requires swap configured on every node)
- Physical CPU required = `total vCPU / CPU overcommit ratio`
- Physical memory required = `total VM memory / memory overcommit ratio`
- Post-overcommit average VM = `physical CPU required / VM count`, `physical memory required / VM count`

STEP 3 – Node Size

- Compute target CPU:memory ratio from post-overcommit average VM size
- Evaluate candidate node sizes and compute effective cores = `physical cores + ((threads − cores) × 0.25)`
- Subtract node overhead: `kube-reserved CPU and memory + 2 cores and 360 MiB OCV fixed overhead`
- Max VMs per node = `min(workload CPU capacity / avg VM CPU, workload memory capacity / avg VM memory)`
- Workload node count = `ceil(total VMs / max VMs per node)`
- Consider blast radius: larger nodes mean fewer nodes but more VMs displaced per failure

STEP 4 – Cluster Architecture

- Memory overcommit via the wasp agent is not compatible with a schedulable control plane, consider HCP or dedicated architectures
- Decide whether to use dedicated infra nodes or co-host infra workloads with VMs
- If co-hosting, add infra workload CPU and memory to per-node consumption

STEP 5 – High Availability

- Decide the number of simultaneous node failures to tolerate `(k)`

- Utilization cap = `(n - k) / n`, normal operating utilization must remain at or below this value
- If utilization exceeds the cap, add nodes until it fits
- Verify both CPU and memory utilization against the cap independently

STEP 6 – Storage, Network, and Special Resources

- Account for storage solution resource overhead (ODF, Portworx, etc.) separately from VM workload sizing
- Plan for network bandwidth: SDN, live migration, storage traffic, and application traffic
- Identify VMs requiring GPUs, SR-IOV, dedicated CPUs, NUMA alignment, etc.
- Check for scheduling constraints such as affinity rules, taints, and tolerations that reduce effective cluster utilization

Appendix: Formulas and reference values

VM memory overhead

```
Memory overhead per VM ≈ (0.002 × requested memory in MiB)
                         + 218 MiB
                         + (8 MiB × number of vCPUs)
                         + (16 MiB × number of graphics devices)
```

Each VM is assumed to have one graphics device unless otherwise specified. Source: OpenShift Virtualization 4.21 documentation.

Effective CPU cores (hyperthreading / SMT)

```
Effective cores = physical cores + ((total threads − physical cores) × 0.25)
```

The 0.25 multiplier is the default used in this document. See Step 3 for guidance on choosing a different value.

Node kube-reserved resources (OCP 4.17+)

CPU, using effective core count as input:

```
Reserved CPU = 0.06 + (0.012 × (effective cores − 1))
               minimum: 0.5 cores
```

Memory:

Node memory	Reserved
First 4 GiB	25%
Next 4 GiB (4–8 GiB)	20%
Next 8 GiB (8–16 GiB)	10%
Next 112 GiB (16–128 GiB)	6%

`Remainder (above 128 GiB)`	`2%`

Source: Red Hat KCS 5843241.

In addition, OpenShift Virtualization adds a fixed overhead of 2 cores and 360 MiB to each compute node.

Failure tolerance utilization cap

To survive the loss of k nodes in an n-node cluster, normal operating utilization must not exceed:

```
Utilization cap = (n − k) / n × 100%
```

T-shirt VM sizes

Size	vCPU	Memory
Micro	1	1 GiB
X-small	1	4 GiB
Small	2	8 GiB
Medium	4	24 GiB
Large	8	48 GiB
X-large	16	64 GiB
XX-large	32	96 GiB

Appendix: Change log

- March 2026, version 3.0.0
  - This is a moderate update that changes the language used in nearly every section. The core process is the same, however there is more explanation and logic provided for the decision making process.
  - Add prerequisites and target audience section, cluster sizing checklist appendix, formulas and reference values appendix, and this change log appendix.
  - Step 1
    - Add VM memory overhead formula from the 4.21 documentation.
    - Added tables to show more precise CPU and memory requirement calculations.
  - Step 2
    - Added explanations of CPU and memory overcommit, including suggestions for how to determine ratios.
  - Step 3
    - Reorganized section.
    - Updated hyperthread / SMT capacity boost to provide more information, suggested ranges, and the default value used is reduced to 25%.
    - Added a table with node size calculations.
  - Step 4
    - Significant changes to this section as a result of schedulable control planes being unsupported with memory overcommit (swap/wasp).
    - Added 2-node with arbiter as potential architecture options.
  - Step 5
    - Memory overcommit with swap removes the need to factor (soft) eviction thresholds into the capacity.
    - Explanation for calculating cluster size based on node failure tolerance has been refactored to be more clear with additional tables for clarity.
  - Steps 6 and 7 had no significant changes.
- June 2026, version 3.1.0
  - Updated step 2's memory overcommit description and math to be more informative and describe how OpenShift's percentage based overcommit is translated into other platforms ratio based logic.