OpenShift Virtualization

Cluster Sizing Guide

Hybrid Platforms Business Unit

version: 2.0.0

February 2025

Legal Notice

Red Hat

Table of Contents

Red Hat

About this Document

Purpose of this document

This document provides guidance, via an example, of how to size an OpenShift cluster to host virtual machine-based workload. The values used in this document represent reasonable values, however it is important to use, as closely as possible, virtual machine sizes that reflect the real expected workload of the cluster. In the absence of known virtual machine sizing data, using the example sizes in this document will suffice. Regardless of the sizing data input and the result of the exercise, it's important to use the monitoring and capacity management tools available to continue to verify and validate all assumptions made, with follow up actions to adjust cluster capacity as needed.

The steps described and used in this document for sizing include:

- Determining the raw capacity needed for the workload
- Factoring overcommitment into the capacity needed
- Identifying node size and the impact to the cluster
- Adjusting node count for cluster architecture
- Further adjusting capacity requirements to account for high availability
- Considerations for special resources

Red Hat

OpenShift Virtualization Cluster Sizing

At a high level, the process is to determine the amount of resources needed for the workload (VM sizes, overhead, burst capacity, failover capacity, etc, while factoring in overcommitment), add that to the amount of resources needed for cluster services (logging, metrics, ACM/ACS if hosted in the same cluster, etc.) and other customer workloads (hosted control planes, other Pods deployed to the hardware, etc.), then find a balance of node size vs node count.

Getting started

To size an OpenShift cluster for virtual machine-hosted applications, we need a few important bits of information:

1. The size of the VMs, specifically CPU and memory.
2. The number of VMs.
3. The amount of overhead for the VMs. This is calculated using a known formula.
4. Desired overcommitment.

With that information we can come up with various scenarios that will both affect, and be affected by, further decisions such as cluster architecture, HA capacity, and special resource requirements.

To begin, we need to understand the desired workload. In the absence of known VM sizing data, we will use “t-shirt” sizing for VMs, however if you have an existing virtualization footprint it is highly recommended to use the real resource utilization data from that environment.

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

Step 1: Determine the raw capacity required for workloads

To set the stage for the remainder of the example, let's identify a desired workload. We will be using the above "t-shirt" size virtual machines, however if VM sizing data from an existing virtualization deployment is available, that should be used.

For this example, let's assume we want an environment that is capable of deploying:

- 1000 small = 2000 vCPU, 8000GiB. Plus, 260 GiB memory overhead.
- 300 medium = 1200 vCPU, 7200GiB. Plus, 92 GiB memory overhead.
- 200 large = 1600 vCPU, 9600GiB. Plus, 77 GiB memory overhead.
- 200 x-large = 3200 vCPU, 12,800GiB. Plus, 96 GiB memory overhead.

This brings us to a total of 1700 VMs using 8000 vCPUs and 37,600 GiB memory, plus 525 GiB overhead. The average VM will have 4.7 vCPUs and use 22.4GiB memory, which is the values we'll use for estimating capacity of the nodes.

Step 2: Factor in overcommitment for CPU and memory

OpenShift Virtualization has supported 10:1 CPU overcommit since it was made generally available with OpenShift 4.5. This means that for every 1 core on the physical host, up to 10 vCPUs can be allocated. This overcommit ratio is enforced by the platform via a resource request in the amount of .1 CPUs (100 millicores) for each vCPU assigned virtual machine.

Beginning with OpenShift 4.17, memory overcommit is supported with OpenShift Virtualization. The default is 150%, or 1.5:1, overcommit, which means that for every 1GiB of memory on the host, 1.5GiB of memory can be assigned to the virtual machines. Furthermore, OpenShift supports the use of technologies such as ballooning and key same-page merging (KSM) to further increase VM memory density on the nodes, however we will not be accounting for those features here.

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.

In this example, we will use values of 4:1 CPU overcommit and 2:1 memory overcommit. To factor this in, we will reduce the average VM size according to these ratios, resulting in values of 1.2 vCPUs and 11.2GiB of memory being used for estimating capacity.

Step 3: Identify node size

The next step is to determine node size by balancing multiple considerations, including density (how many VMs per node), hyperthreading / SMT, avoiding stranded resources, and failure domain. Additionally, you should consider how the Red Hat subscription requirements may affect your choices for CPU core count.

Density

To identify the absolute minimum number of nodes needed, we divide the total number VMs by the maximum number of VMs per node. Current versions of OpenShift support a maximum pod count of 2500, however this is not a reasonable number for virtualization for multiple factors, including but not limited to, kubelet stability. Instead we will use 500 VMs per node as the maximum. Using this maximum VMs-per-node number, this deployment will need a minimum of 4 nodes to be able to host the VM count. This would result in nodes that have `1700 / 4 = 425` VMs each. With the average VM size determined above, we see that the hosts would need capacity for at least `1.2 * 425 = 510` cores and `11.2 * 425 = 4760` GiB memory. Remember, these numbers (1.2 vCPU and 11.2GiB memory) already account for overcommit, so they would represent the real physical resources required by the hosts.

Hyperthreading / SMT

CPUs support multiple threads per core, known as hyperthreading with Intel CPUs or SMT with AMD CPUs, where a single core can run two processes simultaneously. However, this is not a true doubling of the compute power available, rather the extra "thread" is only counted as a fraction of an additional core for capacity purposes. For simplicity, in this exercise we will not repeatedly factor hyperthread / SMT capacity in the calculations. A common choice is to treat the additional thread as providing additional capacity of .5 core each. For a 64 core / 128 thread CPU, this would result in the capacity calculation treating it as having 96 cores. This applies regardless of the socket count, for example a server with 144 cores of capacity could be implemented as a server with two 48 core / 96 thread (`48 cores + (48 threads * .5) = 72 cores`) CPUs or a server with 192 cores could be built using a single 128 core / 256 thread socket or four 32 core / 64 thread sockets.

The calculations used from this point on in the document assume that this has already been accounted for, so they will not be repeated each time.

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. However, it's also important to understand that bare metal OpenShift subscriptions only count the physical cores, not hyperthread / SMT cores, of the CPU. This means that a server with a 128 core / 256 thread layout (across 1-2 sockets), providing 192 cores for capacity purposes using the above assumptions, needs only a single subscription.

Resource isolation

A secondary factor is to attempt to avoid stranding resources by balancing CPU and memory according to requirements. This is as simple as keeping the physical CPU:memory ratio in line with the overall requirement. Factoring in overcommit, we have a requirement for 2000 cores and 19063GiB memory (which includes overhead), resulting in a ratio of 2000:19063. This reduces down to 1:9.53, which is interpreted as "for every physical core on the server, we need 9.53GiB of memory".

This makes it straightforward to see that a server with 192 cores and 2048GiB of memory, a ratio of 1:10.66, which is close to our desired ratio and would result in a modest amount of memory resources being stranded and unable to be used as a result of not enough CPU to schedule workload on the node. Conversely, a server with 108 cores and 2048GiB of memory would have a ratio of 1:19, a significant difference from our desired ratio that would result in large amounts of memory being unused.

Let's assume that we want to use a server with 192 cores. We want to determine the maximum number of VMs that can be hosted on this node by dividing the average per-VM vCPU into this number (`192 / 1.2`), which results in 160 virtual machines. Next we determine the amount of memory that would be needed for those VMs by multiplying that number with the average amount of memory (`160 * 11.2`), which results in 1792GiB of memory needed. Rounding up to 2048GiB of memory, a more common configuration available from hardware vendors, means that our hosts will be CPU constrained. If we were to round down, for example to 1536GiB of memory per node, then the node would only be able to host 137 VMs (`1536 / 11.2`) consuming 164 cores (`137 * 1.2`), which is reinforced that the nodes will be memory constrained by the 1:8 CPU-to-memory ratio being less than the 1:9.53 we desire.

160 VMs per node means our cluster would need 11 servers (actual math is 10.6, but you can't buy fractions of a server). Since we rounded up, we recalculate the number of VMs per server as 155 (`1700 / 11`), which results in each node using 186 cores and 1736GiB of memory.

Node overhead

Thus far the calculation has not taken into account node overhead - we need to have an idea of the node size before we can calculate the overhead. If we allow the nodes to automatically configure system and kube reserved resources, then there is a formula to follow. For a node with 2048GiB of memory, 48GiB will be reserved for the system. A node with 192 cores will have .55 (550 millicores) of CPU reserved. OpenShift Virtualization itself has node overhead as well. This is a negligible 360MiB memory and a flat 2 cores per compute node.

These two overheads combined - approximately 2.55 cores and 48.3GiB memory per node - may affect the size or density of the nodes. When we add these numbers to the 186 cores and 1736GiB memory per node needed by the VMs, we are still below the node capacity which means we do not need to add node(s) to accommodate the overhead.

As it stands now, our cluster is 11 servers of 192 cores and 2048GiB memory each. These servers will host 155 (1700 / 11) virtual machines each, totaling 189 cores and 1784GiB of memory used. This results in 98% CPU and 87% memory allocation for each node.

Failure domain

Failure domain refers to the amount of resources affected by a node failure. For a cluster with 11 nodes, a single node failing represents 9% of total cluster capacity and, approximately, 160 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.

With the goal of not wasting resources, increasing or decreasing node size, ideally inline with the desired CPU:memory ratio, would result in a larger or smaller, respectively, failure domain. For example, a node with 336 cores (e.g. 4 x sockets with 56 cores each + 50% thread capacity) and 3072GiB memory (1:9.14 ratio), when following the above steps, results in a cluster of just 7 nodes (memory constrained) hosting 242 VMs each. The failure of a single node represents 14% of cluster capacity. Conversely, nodes of 72 cores and 768GiB memory would create a cluster of 28 nodes, each one being just 3.6% of cluster capacity.

Let's move forward that 11 nodes represents an acceptable failure domain for this sizing exercise.

Step 4: Adjust for cluster architecture

There are three architecture options:

- Single node OpenShift
- Schedulable control plane
- Dedicated control plane

SNO has some limitations, and sizing is relatively straightforward, so that we can ignore it for this exercise.

Schedulable and dedicated control plane architectures are two sides of the same coin, but we need to make some additional decisions to get enough information to continue sizing the cluster.

- If using schedulable control plane nodes, be sure to include those capacity requirements in the total capacity needed.
- Do we want homogenous node sizes, or are we willing to use a different node size for the control plane and infra?
    - If yes, consider having schedulable control plane nodes that also host infra workload. Depending on the node size/capacity, these nodes may still have capacity for virtual machines.
- How many control plane nodes will be used? Three is the standard, however a five-node control plane architecture may be worth considering if hardware replacement, in the event of a failure, can take an extended amount of time.
- Will this cluster use a hosted control plane? If yes, then that capacity does not need to be accounted for in this exercise.
- Which of the infra-qualified add-on features, e.g. logging, etc., will be hosted by the cluster?
    - If the cluster does not have dedicated infra (or combined infra + control plane) nodes, this capacity will need to be added to the overall requirement.
- Is the customer planning on using features, such as pipelines and/or GitOps, which will increase the API server, and other control plane component, utilization?
    - Each of these places an additional burden on the control plane components. This may affect the size of those nodes if they're dedicated or how many VMs can be hosted on the control plane nodes if they're schedulable.

In step 3 we determined that we need 11 nodes to host the virtual machines and associated overhead. Let's make some assumptions based on the above information. Specifically, we are

going to 1) use schedulable control plane nodes and 2) infrastructure workload will not have dedicated nodes.

For the sake of simplicity, let's assume that the combined resource requirements for control plane and infra workloads total approximately 40 cores and 256GiB memory. The total workload resource requirements for the cluster are the sum of the VM requirements and control plane / infra, which totals `2000 + 40 = 2040` cores and `19063 + 256 = 19319` GiB memory. These values fit within the capacity of the 11 cluster members, so no additional nodes are required.

> **Important:** The number used for control plane + infra workload resources is not based on validated numbers. 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

We factored in the size of the failure domain above, however we have not taken into account how to accommodate that capacity in the event of a failure scenario - or even when taking nodes offline, for example, to perform cluster updates and upgrades. With an 11 node cluster, each node represents 9% of cluster capacity, so we need to accommodate for one node's worth of capacity to be lost, plus that capacity being rescheduled across the surviving nodes. This can be calculated as `9 * #_of_node_failures` worth of capacity to be accounted for in the event of failure.

Explaining this differently, a cluster of 10 functioning nodes has 100% capacity. If a node fails, the total cluster capacity is reduced to 90% of original. And, we need to reschedule the workload that was previously hosted on that node to the surviving members, which represents 11% of the remaining cluster capacity (the failed node represents 10% of a 10-node cluster's capacity, but 1 node is 11% of a 9-node cluster).

To illustrate this with an example, let's say that 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, I now have 9 nodes remaining to host the 100 virtual machines, which still need 800 units of capacity in my cluster which now has only 900 available. My cluster utilization went from 80% of 10 nodes to 89% of 9 nodes.

If my 10-node cluster hosting 100 virtual machines is running above 90% capacity, meaning more than 900 of 1000 units of capacity are being used, then I can no longer tolerate a node failure. My 9 node / 900 capacity unit cluster cannot restart virtual machines totaling more than 900 units of capacity.

If we expand this to two node failures, my 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. This continues to scale linearly.

For our cluster:

- Tolerating one node failing would limit the cluster to 91% maximum utilization.
- Tolerating two nodes failing would limit the cluster to 82%.
- Tolerating three nodes failing would limit the cluster to 73%.

Furthermore, we want to avoid resource contention scenarios, such as out-of-memory conditions resulting in VMs being swapped or terminated. When to trigger evictions will depend on a number of factors, most importantly we want to set the value low enough where soft evictions will have time to happen before hard evictions and/or any actions are taken by the OOM_kill process. A secondary consideration is that the soft eviction will also become the threshold where a node's workload triggers rescheduling, which can act as a rudimentary way for rebalancing resources as well.

> **Note:** When memory overcommit is being used, the node eviction threshold is automatically adjusted and managed via the wasp-agent, which factors in the soft eviction threshold.

Depending on the size of the nodes, the size of the VMs, and the length of time to live migrate virtual machines, it may make sense to have the soft eviction threshold set anywhere between 85% - 99%. Since our servers have a substantial amount of memory it makes sense to choose a higher value. Using a 96% memory utilization threshold on servers with 2048GiB of memory would trigger soft evictions (i.e. live migrations) for VMs when the host has approximately 82GiB of free memory remaining. This may or may not be enough for the workload and would need to be monitored and adjusted over time.

For the sake of sizing, we can now assume that for a 11-node cluster, where we want to tolerate two nodes failing with a soft eviction threshold of 4% remaining capacity, we can calculate that we are increasing overall resource requirements by 22% (18% for node failover capacity, 4% for eviction) or, put differently, we are limiting the maximum utilization of the nodes to 78%. This

means we would want to increase the node count to accommodate the reduced capacity, `11 / 78% = 14.1`. Rounded up, we need 15 nodes total. Redoing the math from step 3, when all 15 nodes are healthy they will each host `1700 / 15 = 113` virtual machines, with resource utilization of 135 cores (70%) and 1512GiB memory (74%) 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

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 ODF, Portworx, IBM Fusion and other Kubernetes-native storage solutions.
- 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?

These can be difficult to accommodate in a generic sizing exercise like this. Instead, it may be helpful to size them separately in an exercise dedicated to those workloads. You may decide it is better to have one or more dedicated clusters for those workloads or use features such as taints and node selectors to keep cluster count to a minimum.

(Optional) Step 7: Disaster recovery

See the OpenShift Virtualization Disaster Recovery Guide for strategies and methods to recover virtual machines at the destination site. For workloads which are identified to be recovered, follow the above steps for the cluster at the destination site with the appropriate considerations. If you intend to use an existing cluster at the destination site, be sure to include the workload for both the intended and recovery VMs in capacity calculations.

Final result

We will not include the considerations in steps 6 or 7 for this exercise since they depend on organization-specific decisions and implementation. The final result, as shown in step 5 based on the assumptions and decisions made, is a 15 node cluster composed of nodes that have 192 cores and 2048GiB memory.

To illustrate how different node sizes affect the cluster geometry, the table below shows additional options for larger and smaller physical node sizes.

Node resources	Nodes required for workload	Node failure toleration	Node resource threshold	Cluster node count
24 cores / 256GiB memory	85	4	90%	100
48 cores / 512GiB memory	43	3	90%	52
72 cores / 768GiB memory	29	3	92%	36
108 cores / 1024GiB memory	19	2	92%	24
144 cores / 1536GiB memory	15	2	94%	19
192 cores / 2048GiB memory	11	2	96%	15
384 cores / 4096GiB memory	6	1	97%	8
576 cores / 8192GiB memory	4	1	99%	6