OpenShift Virtualization

Reference Implementation Guide

Hybrid Platforms Business Unit

version: 1.0.2

September 2024

Legal Notice

Red Hat

Table of Contents

About this Document

Purpose of this document

This document provides implementation guidelines and a sample reference architecture for deploying Red Hat OpenShift as a platform for virtualization workloads using OpenShift Virtualization. The design guidelines and architecture described provide a target platform to host workloads that may currently reside on one of the following platforms:

- VMware vSphere Foundation
- Red Hat Virtualization
- OpenStack

This document is a supplement to product documentation providing opinionated suggestions and guidance for configuring OpenShift for hosting virtual machines. It suggests several cluster architectures for varying capacity needs, provides an overview of the technology in use, provides design considerations, and provides prescriptive guidance for implementation. The guidelines in this document do not define a required configuration for support, nor are they exhaustive. Before implementing, consider the needs and requirements of your virtualized applications and choose the configuration that best meets those needs.

Target Use Cases

OpenShift Virtualization helps customers optimize their data center footprint for today's and tomorrow's applications. It provides modern infrastructure for enterprise VMs that increases efficiency through automation, manageability, and scalability. OpenShift Virtualization delivers the performance, scale, and stability of Kernel-based virtual machines (KVM), deployed and managed as part of a modern application platform.

OpenShift Virtualization enables businesses to accelerate infrastructure modernization by bringing modern virtualization infrastructure, via OpenShift, to support existing virtual machines.

Customers will see value in several ways:

Replatforming from Legacy Virtualization Platforms

OpenShift is a trusted, comprehensive, and consistent platform designed to scale with the needs of today's virtualization workloads. As customers move to OpenShift Virtualization from a legacy platform, they will see immediate benefits as the management of virtual machines shifts to a cloud-native paradigm via OpenShift's Kubernetes foundation.

Modernize Today For Innovation Tomorrow

With OpenShift Virtualization, organizations can modernize their virtualization platform and benefit from modern management principles integrated into OpenShift. By adding existing VMs to mixed applications, customers can run their virtualization machines and containers side by side.

Consistency Of Management

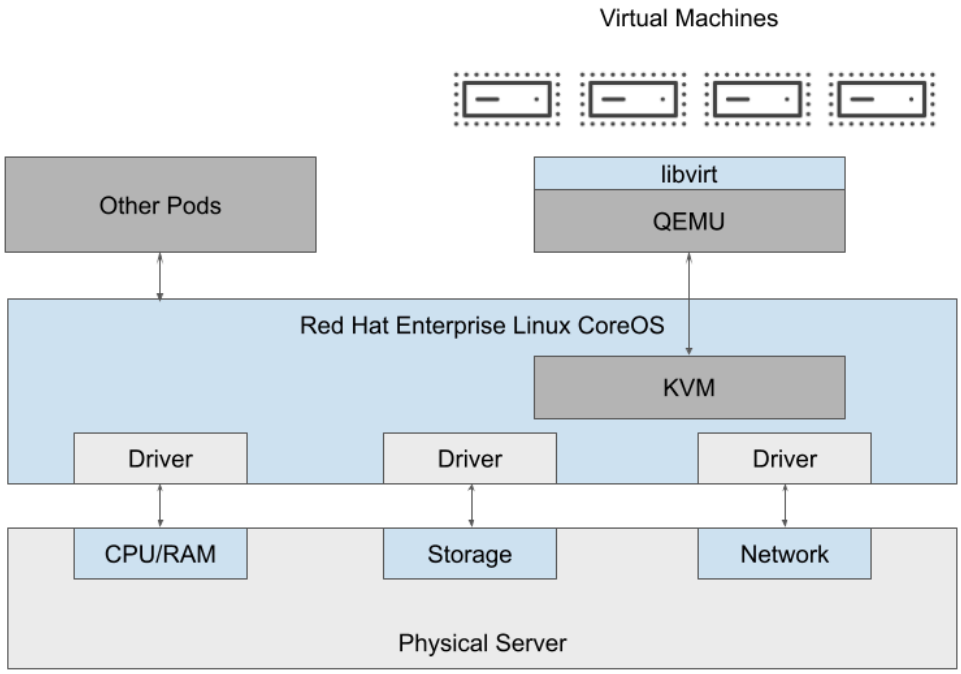
From the OpenShift console, customers can manage VMs, containers, and serverless in one place. Consistent management of workloads will drive better alignment between VM admins and platform engineering teams.

Increased Operational Efficiency

The self-service options in OpenShift help teams improve their time to production. Specifically, customers can streamline deployments of their application and development stacks by integrating VMs with development pipelines via the OpenShift API and integrating the CI/CD pipelines directly into OpenShift deployments.

Technology Overview

OpenShift Virtualization uses KVM, the Linux kernel hypervisor, to bring virtual machines as native objects to OpenShift, with the features and capabilities of OpenShift supporting cluster management and scalability, along with virtual machine consumption. KVM is a core component of the Red Hat Enterprise Linux kernel and has been used in production for 15+ years in other hypervisor products, such as Red Hat Virtualization, Red Hat OpenStack Platform, and Red Hat Enterprise Linux. The KVM used by OpenShift is the same KVM, QEMU, and libvirt used by those platforms; OpenShift is a robust, stable, and scalable type-1 hypervisor for virtualization workloads.



Virtual machines are deployed and managed using the same scheduler that is used for non-virtual machine workloads. Native features like host affinity, resource awareness, load balancing, and high availability are all inherited from OpenShift features for non-virtual machine workloads.

Virtual machines connected to the cluster software-defined network (SDN) are accessible using standard Kubernetes methods, including Services and Routes, while also being subject to controlling traffic using network policy and ingress/egress configuration. This provides robust methods for enabling access to virtual machines from inside and outside the cluster, along with platform-level security controls for managing access to VM-hosted applications.

OpenShift Virtualization has a number of deployment options available, including self-managed bare metal hosts on-premises, or in the cloud on AWS using either managed ROSA clusters or self-managed clusters.

Reference design and architecture

We recommend a modular architecture with a structured, standardized approach for designing and implementing Red Hat OpenShift Virtualization. This framework guides customers, sales, and services teams through a standardized approach that facilitates scalability, security, and compliance with industry standards. By leveraging these design guidelines, organizations can accelerate their digital transformation initiatives, improve system interoperability, and achieve more efficient and effective IT operations. It serves as a blueprint, offering best practices, patterns, and guidelines that address current and future technological challenges, enabling businesses to innovate and adapt in a rapidly changing digital landscape.

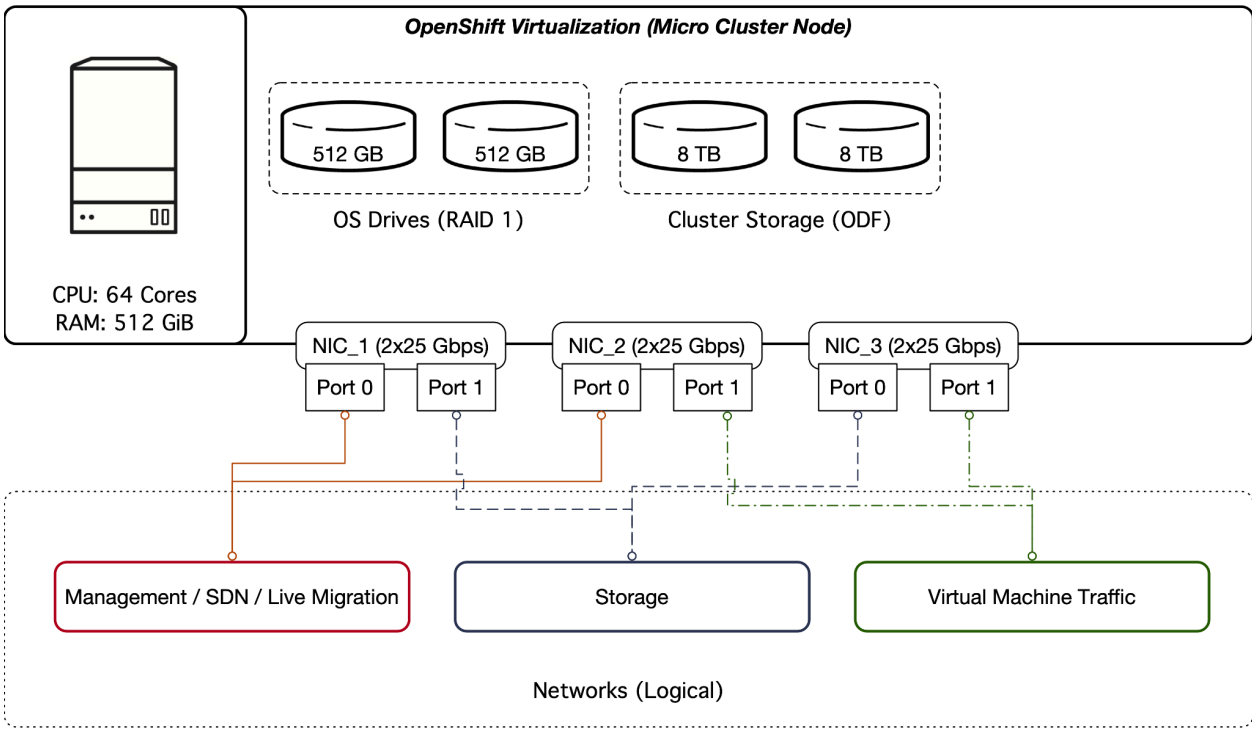
Physical Cluster Design

In the absence of known sizing for compute and storage resources, Red Hat recommends using a modular approach to node sizing based on the expected initial capacity requirements, with the ability to scale up to 100+ hypervisor nodes in a single OpenShift deployment. OpenShift Virtualization supports many different node and cluster configurations. In the absence of known sizing requirements, the sizing below represents a starting point for an OpenShift deployment; however, node sizes and configurations should be tailored to meet your requirements as needed.

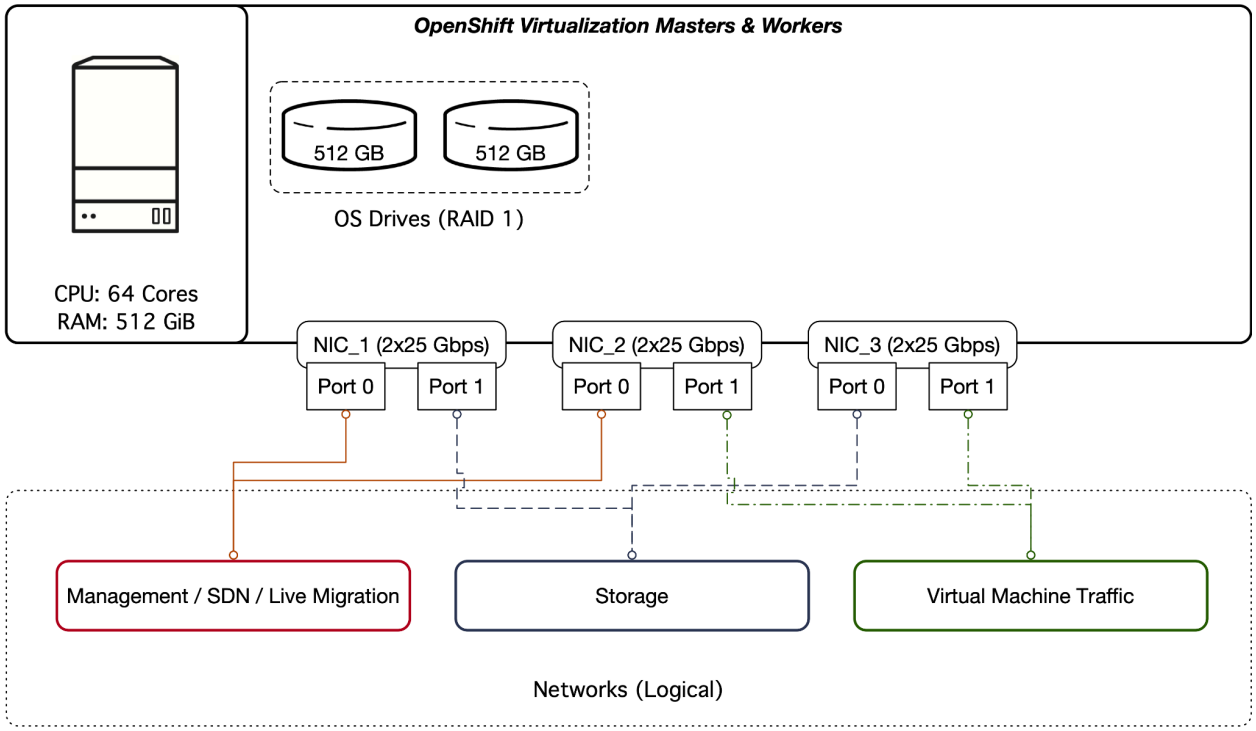
Node Design per Architecture

Here is a representation of the physical node characteristics, including the storage and network layout of the nodes recommended for this design depending on the architecture deployed:

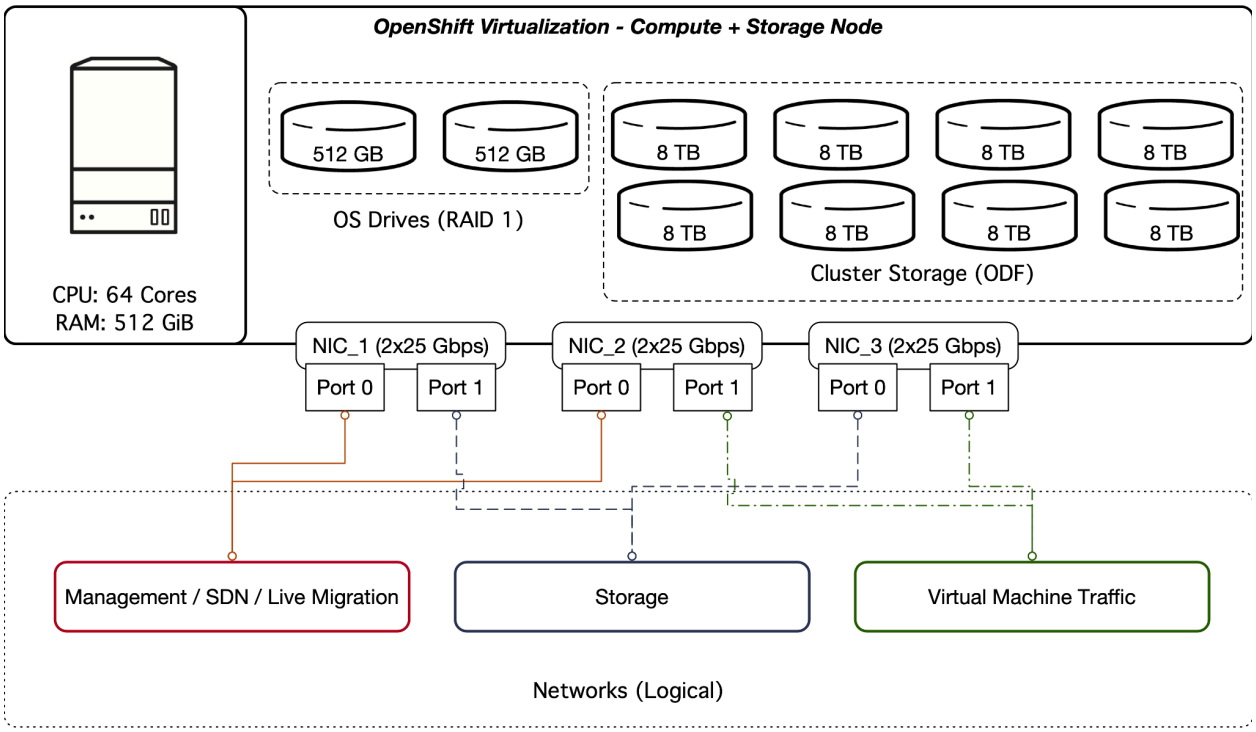
- Micro Architecture Cluster Node (hyper-converged):



- Control + Compute (master + worker) node:



- Compute + Storage (worker with ODF) node:

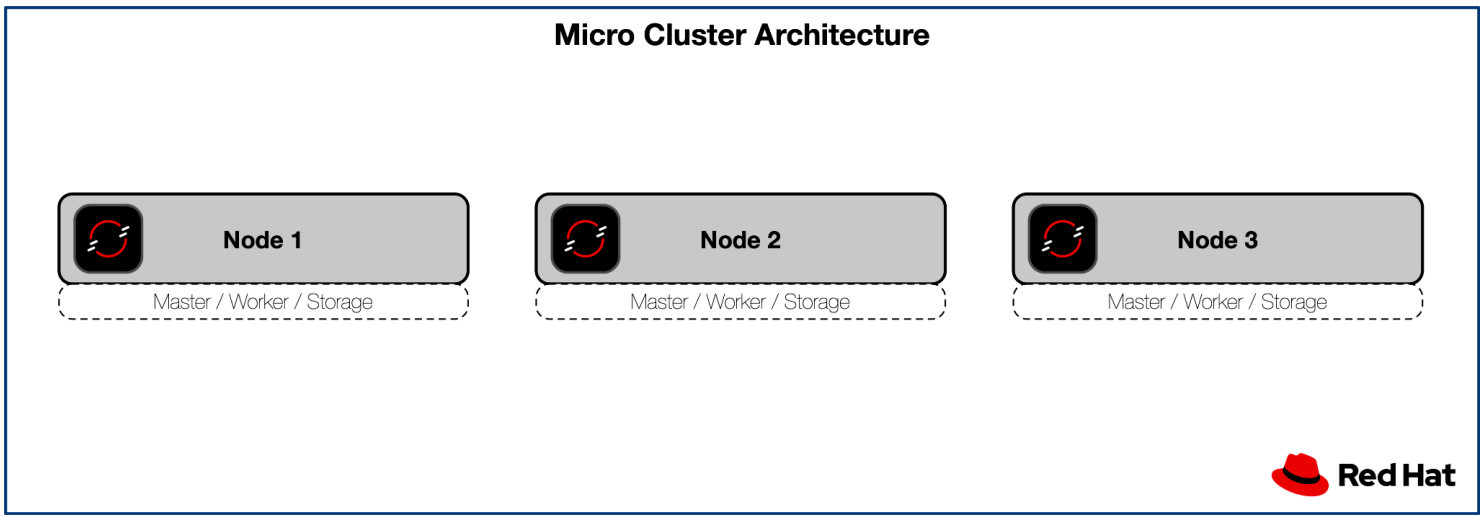


Micro

This cluster size is comprised of:

- A three (3) node OpenShift cluster
    - Control plane running on three (3) nodes
    - Storage and compute running on all three (3) nodes with compute
    - Example node size:
        - CPU: 64 cores
        - Memory: 512GiB
        - OS Disks: 2x512GB (RAID1)
        - Cluster Storage (ODF): 2x 8TB NVMe/SSD disks
        - Network: 6x 25Gb NICs
    - The resulting cluster has 336 vCPU (with 4:1 overcommit), 826GiB memory, and 10.4TB storage available for workloads.
        - 34% of compute and memory capacity is reserved for HA
        - 10% of compute capacity is reserved for spare/burst
        - The suggested storage capacity above reflects 65% of the capacity used

The high-level design for the micro form factor cluster would look like the following:



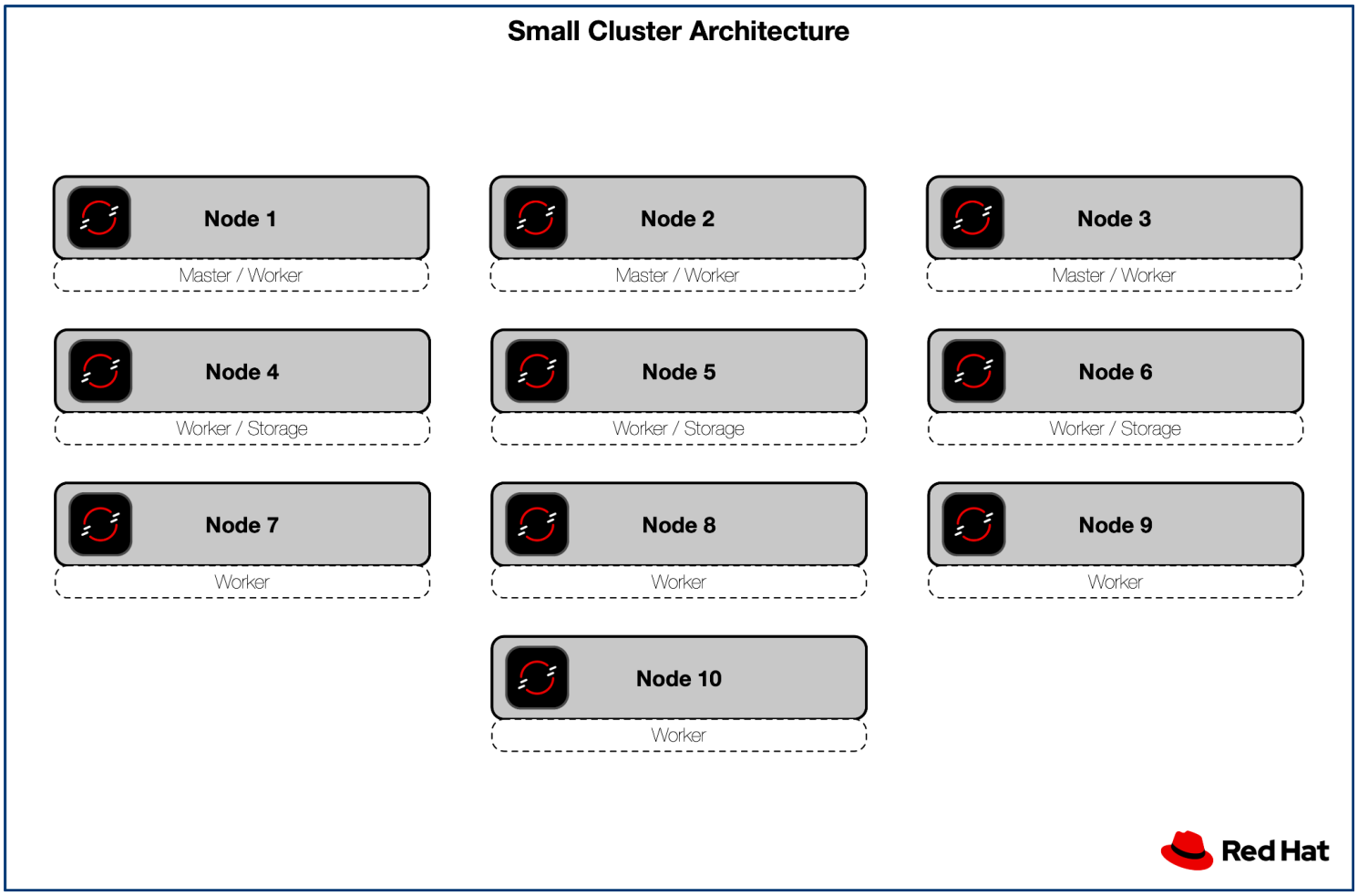
Small

This cluster size is comprised of:

- Ten (10) node OpenShift cluster
    - Control plane (masters) + compute (workers) running on three (3) nodes
    - Storage running on three (3) nodes with compute
    - Additional four (4) nodes of "just" compute
    - Example node size:
        - CPU: 64 cores
        - Memory: 512GiB
        - OS Disks: 2x512GB (RAID1)
        - Cluster Storage (ODF): 8x 8TB NVMe/SSD disks
        - Network: 6x 25Gb NICs
    - The resulting cluster has 1532 vCPU, 3496GiB memory, and 41.6 TB storage available for workloads.
        - 20% of compute and memory capacity is reserved for HA
        - 10% of compute capacity is reserved for spare/burst
        - The suggested storage capacity above reflects 65% of the capacity used

The resulting cluster has three node types (compute, compute + storage, compute + control plane) which contribute capacity for hosting virtual machines in addition to their shared purpose (ODF or control plane).

Here is a high-level design for the small form factor cluster:



Medium

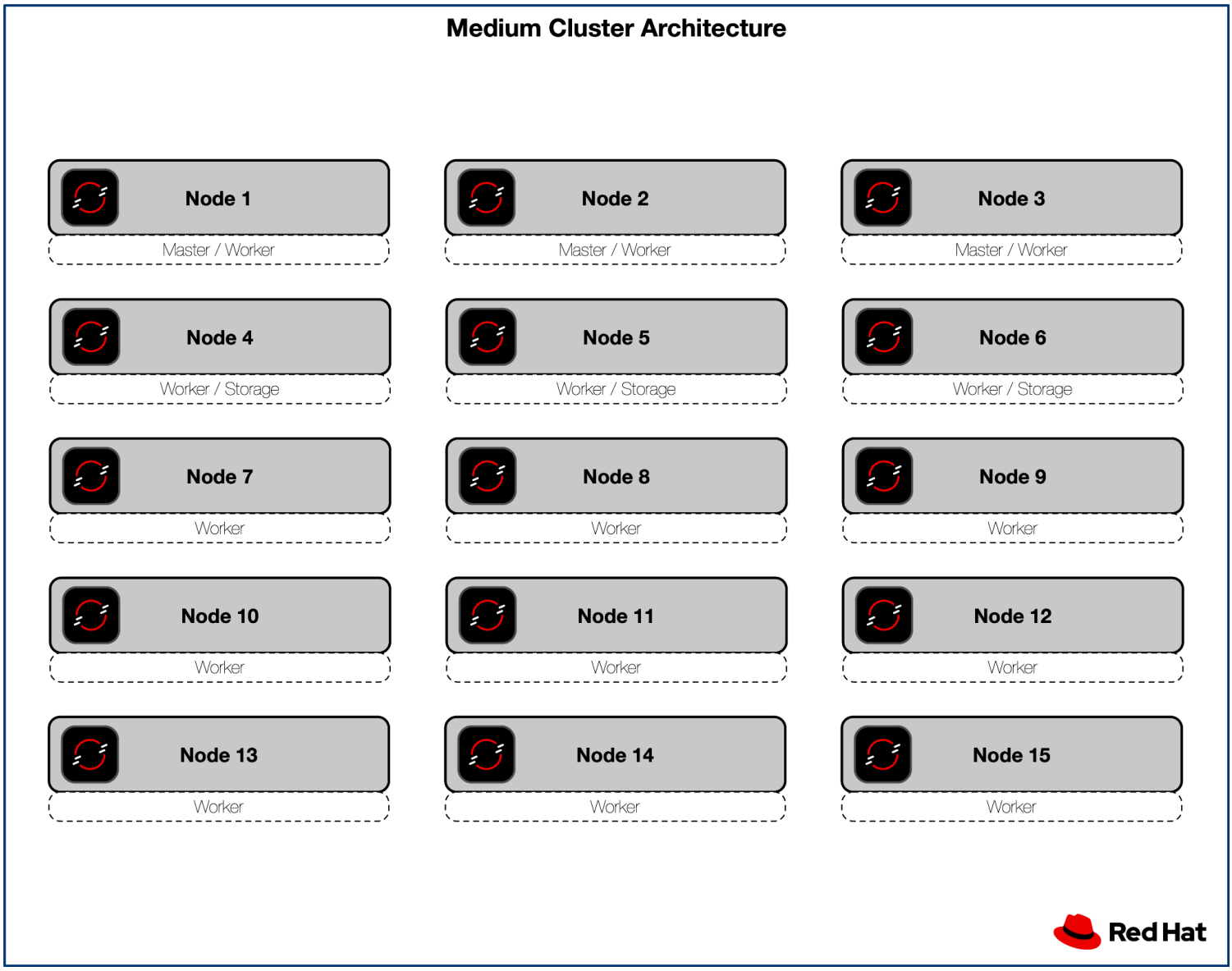
This cluster size is comprised of:

- Fifteen (15) node OpenShift cluster
    - Control plane + compute running on three (3) nodes
    - Storage running on three (3) nodes with compute
    - Additional nine (9) nodes of "just" compute
    - Example node size:
        - CPU: 64 cores
        - Memory: 512GiB
        - OS Disks: 2x512GB (RAID1)
        - Cluster Storage (ODF): 8x 8TB NVMe/SSD disks
        - Network: 6x 25Gb NICs

- The resulting cluster has 2345 vCPU, 5177GiB memory, and 41.6 TB storage available for workloads.
  - 10% of compute and memory capacity is reserved for HA
  - 10% of compute capacity is reserved for spare/burst
  - The suggested storage capacity above reflects 65% of the capacity used

The resulting cluster has three node types (compute, compute + storage, compute + control plane) which contribute capacity for hosting virtual machines in addition to their shared purpose (ODF or control plane).

Here is a high-level design for the medium form factor cluster:



Large

- Scale the medium design further, taking into consideration that we may consider non-converged storage based on use cases and cluster architectures:
    - Control plane running on three (3) nodes
    - Storage running on a minimum of 4 nodes with compute
    - Compute running on all nodes
    - Scale storage nodes in increments of four, which aligns with ODF entitlements of 256TB raw storage
    - Scale compute nodes in increments of one (1)

We do recommend that when deploying OpenShift Virtualization, failure domains are also defined at the physical architecture layer when possible to provide better resiliency. Topology labels can be applied to nodes and used by VMs and containerized applications to spread across failure domains. A few examples of spreading across failure domains are:

- Use a multi-rack topology (at least 3), distributing the roles across the multiple racks
- Each rack is a failure domain
- Deploy a CLOS Leaf-Spine architecture, where each rack is contained within a Leaf

For more information, check the following references:

1. https://kubernetes.io/docs/reference/labels-annotations-taints/#topologykubernetesioregion

Reference implementation

Deploying and configuring OpenShift features to support virtual machines and applications

Using the above cluster geometry as a starting point, the OpenShift cluster(s) need to have features and functions configured to provide capabilities for supporting the deployment and management of virtual machines, along with value-add capabilities for virtualized applications. This section will describe the recommended OpenShift configuration for hosting virtual machines.

Cluster deployment configuration

Installer Provisioned Infrastructure

This architecture uses the installer provisioned infrastructure (IPI) method for deploying the cluster. The resulting cluster will have the `baremetal` cloud provider configured and will be able to make use of it for adding and managing nodes. For more information on IPI-based bare metal installations, please see the documentation here.

For the suggested cluster designs in this reference architecture, the control plane should be schedulable. When instantiating the cluster, deploy a three node "compact" cluster, which will mark the control plane nodes as schedulable by default. After deployment, join additional compute and storage nodes as needed, using distinct MachineSets for each role, e.g. storage and compute.

Red Hat Enterprise Linux CoreOS (RHCOS)

OpenShift Virtualization requires Red Hat Enterprise Linux CoreOS (RHCOS) compute nodes. Even though it is possible to deploy and use Red Hat Enterprise Linux (RHEL) compute nodes, they are incompatible with OpenShift Virtualization. RHCOS is based on RHEL and designed to run containers with an integrated CRI-O runtime and dedicated container tools. The operating system uses rpm-ostree to facilitate being managed through the Machine Config Operator to deploy operating system updates and apply configuration declaratively.

To customize RHCOS, for example, to add drivers or other third-party OS-level tools, the RHCOS image layering feature must be used to modify the operating system and add the appropriate packages.

For more information on Red Hat Enterprise Linux CoreOS, see the documentation.

Bonded NICs for management and SDN

The initial bond interface, consisting of two adapters bonded together with an IP address on the machine network specified and configured at install time, is used for the SDN, management traffic between the node and the control plane (and administrator access), and live migration traffic. During installation, use the host network interface configuration options to configure the bond and set the IP address needed.

Post-install configuration

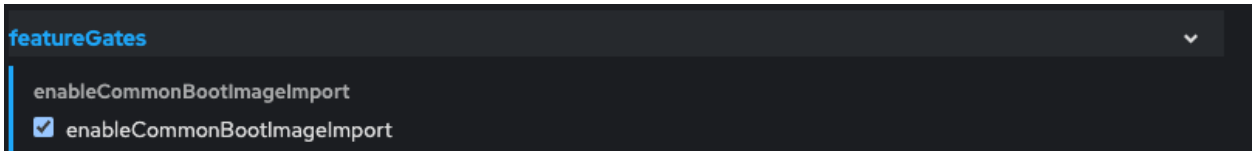
After the OpenShift cluster has been deployed, with additional nodes joined for compute and storage roles, additional cluster configuration is needed to adjust and optimize the configuration for virtual machines.

OpenShift Virtualization Operator configuration

OpenShift Virtualization is deployed and configured in the cluster using OperatorHub. Once installed, there are several configuration options to set.

Follow the documentation for deploying the Operator and creating an instance of the HyperConverged resource. The below bullets represent common decisions and configuration parameters that are not default.

- Decide whether to download the default, Red Hat provided, operating system images. The Operator will automatically retrieve and make available cloud images for RHEL, Fedora, and CentOS when a default storage class is configured. If you do not intend to use these images, it is recommended to disable automatically retrieving them.



- Configure the default CPU model to the lowest common denominator CPU model in the cluster. These values are determined by the CPU vendor and the highest, or lowest, CPU model available in the OpenShift cluster. For AMD CPUs, a value of `EPYC` is recommended, for Intel CPUs a value equivalent to the generation name, e.g. `Cascadelake-Server`, is recommended.

- If you do not want to use the default storage class for storing VM state related to, among other things, vTPM, set the desired storage class when deploying OpenShift Virtualization. This storage class must support RWX access mode.

Machine and node configuration

These tasks represent configurations applied to the cluster nodes after the cluster is deployed.

- Configure time synchronization to avoid clock drift and issues related to inaccurate/inconsistent time, such as certificates being incorrectly marked invalid.

- Configure node health checks and fence agents remediation for reduced HA recovery time in the event of node failure.

  The Node Health Check Operator and Fence Agents Remediation Operator are deployed from OperatorHub in the web console. Once installed an instance of the FenceAgentsRemediationTemplate must be created to provide BMC connection information for the nodes. This is the same information that was used when deploying the cluster and joining nodes using a MachineSet. Finally, create an instance of the NodeHealthCheck to define appropriate timeout values for your desired VM recovery time. Refer to the chart in this KCS for details on how different values affect the recovery times.

Additional dedicated network interfaces for traffic types

The suggested hardware configuration used in this reference architecture has six network adapters configured in three pairs of two LACP bonds. The following is a sample NMstate configuration making use of two adapters on the host to create a bonded interface in the LACP (802.3ad) run mode.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  annotations:
    description: a bond for VM traffic and VLANs
  name: bonding-policy
spec:
  desiredState:
    interfaces:
      - link-aggregation:
          mode: 802.3ad
          port:
            - enp6s0f0
            - enp6s0f1
        name: bond1
        state: up
        type: bond
```

The bonds are intended to be used for isolating network traffic for different purposes. This provides the advantage of avoiding noisy neighbor scenarios for some interfaces that may have

a large impact, for example a backup for a virtual machine consuming significant network throughput impacting ODF or etcd traffic on a shared interface.

- Management, SDN, and live migration - this is the interface configured during installation, with an IP on the machine network CIDR. OpenShift will configure the SDN on this interface, and it is the interface expected to be used for node-to-control plane communication.

  Additionally, by default, OpenShift Virtualization uses this interface for live migration traffic. This has the benefit of the traffic being encrypted in transit; however, if SDN throughput is constrained, or other workloads would be impacted, then using a different network, such as one shared with the IP-base storage, for live migration traffic.

- IP-based storage - this is the interface which will be used for two purposes in this reference architecture. Using additional isolation, such as VLANs, on this interface is possible, but not required.

  First, it is the interface expected to be used for mounting iSCSI, NFS, and RBD volumes used by virtual machine disks. If the node IPs and the storage system are not in the same broadcast domain (i.e. L2 adjacency) then additional route rules may need to be created using NMstate to ensure that traffic to the storage system utilizes this interface.

  Second, for deployments using ODF, this is the interface to be used for ODF replication traffic.

- Virtual machine traffic - this interface will be configured with an OVS bridge to attach virtual machines. For each VLAN used by virtual machines, an OVN secondary network `NetworkAttachmentDefinition` is created using the localnet topology and a VLAN identifier. `NetworkAttachmentDefinitions` are created in the `default` namespace for use by all other namespaces, or in specific namespaces to provide RBAC controls for network connectivity.

  An example configuration for VM network connectivity is below, note that the bond configuration should be a part of the same NodeNetworkConfigurationPolicy to ensure they are configured together.

```
    apiVersion: nmstate.io/v1
```

```
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ovs-br1-vlan-trunk
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ''
  desiredState:
    interfaces:
    - name: ovs-br1
      description: |-
        A dedicated OVS bridge with bond2 as a port
        allowing all VLANs and untagged traffic
      type: ovs-bridge
      state: up
      bridge:
        allow-extra-patch-ports: true
        options:
          stp: false
        port:
        - name: bond2
    ovn:
      bridge-mappings:
      - localnet: vlan-2024
        bridge: ovs-br1
        state: present
      - localnet: vlan-1993
        bridge: ovs-br1
        state: present
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    description: VLAN 2024 connection for VMs
  name: vlan-2024
  namespace: default
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "vlan-2024",
      "type": "ovn-k8s-cni-overlay",
```

```
        "topology": "localnet",
        "netAttachDefName": "default/vlan-2024",
        "vlanID": 2024,
        "ipam": {}
      }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    description: VLAN 1993 connection for VMs
  name: vlan-1993
  namespace: default
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "vlan-1993",
      "type": "ovn-k8s-cni-overlay",
      "topology": "localnet",
      "netAttachDefName": "default/vlan-1993",
      "vlanID": 1993,
      "ipam": {}
    }
```

Kubelet configuration

The following additional configuration for kubelet is applied after the cluster is deployed.

- Increase kubelet `kubeAPIBurst` to 200 and `kubeAPIQPS` to 100. Adjusting these values up from the default of 100 and 50, respectively, accommodates bulk object creation on the nodes. The lower values are useful on clusters with smaller nodes to keep API server resource utilization reasonable, however with larger nodes this is not an issue.
- Set the `maxPods` per node to 500. By default, OpenShift sets the maximum Pods per node to 250. This value affects not just the Pods running core OpenShift services and node functions but also virtual machines. For large virtualization nodes that can host many virtual machines, this value is likely too small. If you're using very large nodes and may have more than 500 VMs and Pods on a node, this value can be increased beyond 500, however you will also need to adjust the size of the cluster network's host prefix when deploying the cluster.

- Disable `nodeStatusMaxImage`. The scheduler factors both the count of container images and which container images are on a host when deciding where to place a Pod or virtual machine. For large nodes with many different Pods and VMs, this can lead to unnecessary and undesired behavior. Disabling the imageLocality scheduler plugin by setting `nodeStatusMaxImage` to `-1` facilitates balanced scheduling across cluster nodes, avoiding scenarios where VMs are scheduled to the same host as a result of the image already being present vs factoring in other resource availability.
- Set dynamic resource allocation for kubelet. The default CPU and memory reservation for kubelet is very small and not appropriate for nodes with large amounts of resources, such as hypervisor hosts. Setting dynamic resource allocation will set the values for system reserved CPU and memory according to the total amount of resources on the node. This prevents kubelet from starving for resources when the node has high numbers of Pods and virtual machines running.
- Configure CPU manager to enable dedicated resources for virtual machines to be assigned. Without CPU manager, virtual machines using dedicated CPU scheduling, such as those configured with the cx instance type, cannot be scheduled.
- Configure soft eviction thresholds. Configuring soft eviction is valuable for several reasons, however the most important is that it sets the upper boundary for memory utilization on the nodes before the virtual machines are attempted to be moved to other hosts in the cluster. This value should be set for a reasonable value according to the approximate maximum amount of memory utilization you want on the node, however if all nodes are exceeding this value then workload will not be rescheduled. Depending on the amount of memory in your hosts, this could be between 90-95%. For the suggested node size in this architecture, a value of 90% is used.

  Additional eviction thresholds for local storage utilization can be configured based on needs, in particular if the disks used by RHCOS are smaller (less than 512GiB) or you intend to deploy ephemeral virtual machines with disks stored in container images.

Below is an example kubelet configuration to apply the above suggested configuration to compute nodes in the cluster. If you have hypervisor hosts that do not have the “worker” label, an additional selector may be needed.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-virt-values
```

```
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
  kubeletConfig:
    autoSizingReserved: true
    maxPods: 500
    nodeStatusMaxImages: -1
    kubeAPIBurst: 200
    kubeAPIQPS: 100
    evictionSoft:
      memory.available: "50Gi"
    evictionSoftGracePeriod:
      memory.available: "5m"
    evictionPressureTransitionPeriod: 0s
```

OpenShift Data Foundation configuration

This reference architecture utilizes ODF as the storage backed for virtual machine disks. Storage is modular and additional CSI providers can be added to the cluster to provide both ODF and third-party storage from other vendors. Furthermore, ODF is not required if you choose to exclusively use RWX storage from other vendors. For non-ODF storage requirements, see the storage section below.

- Set global, and virtualization specific, default storage classes. The global default storage class is used as the default for PVCs when a specific storage class is not requested. For this reference architecture, when using ODF for the storage, the suggested global default is `ocs-storagecluster-ceph-rbd`. The virtualization specific default storage class is used when creating virtual machine disks. The default when using ODF should be `ocs-storagecluster-ceph-rbd-virtualization`, unless creating customized storage classes.
- Create additional storage classes as needed to customize the storage features, however ensure that the `parameters.mapOptions` value `krbd:rxbounce` is set for any ODF storage class used for VM disks.

Multi-network policy activation

To support network policy for VMs connected to L2 networks, the feature needs to be enabled on the Operator. When using network policy to control ingress/egress traffic for virtual machines, use the MultiNetworkPolicy object to define rules for access.

Additional Operator configuration

The below Operators are suggested to be deployed and configured in the cluster to provide additional functionality.

- Node maintenance Operator - Simplifies the process of cordoning and draining nodes in the cluster for maintenance purposes.
- OpenShift logging - Collects the logs of cluster nodes, Pods, and VMs (but not the guest OS logs) into a single place for searching and troubleshooting.
- MetalLB - Provides the ability to expose Pod and VM-based applications using an L2 or L3 (BGP)-based load balancer hosted by OpenShift. This is useful for providing external access to one or more VMs connected to the SDN that are hosting an application using a non-HTTP(s) (and not using ports 80 or 443) application.
- Migration Toolkit for Virtualization - In addition to importing virtual machines from Red Hat Virtualization, Red Hat OpenStack, and VMware vSphere to OpenShift, the Migration Toolkit for Virtualization also imports VMs from OVA format and between OpenShift clusters. Additionally, virtual machines from other OpenShift Virtualization clusters can be migrated to the current cluster using the migration toolkit.
- Migration Toolkit for Containers - Used to migrate VM disks (PVCs) between storage classes.
- Compliance Operator - For organizations with compliance, e.g. HIPAA or DISA STIG, requirements, the Compliance Operator reports, and optionally applies, configuration to the cluster to meet these standards.
- Kube Descheduler Operator - The descheduler takes action to (re)schedule workload on the cluster nodes when conditions are met. For virtualization, this will take action when nodes are below a utilization threshold, 20% by default, to reschedule VMs from highly utilized nodes, over 50% by default, with the intention of balancing overall node utilization. However, the profile is currently in tech preview.
- NUMA resources Operator - As the name implies, this Operator is used when NUMA aware workloads are deployed to the cluster. For VMs with large memory requirements or applications that can benefit from NUMA-aware scheduling and

utilization of resources, deploying this Operator and assigning the appropriate secondary scheduler informs the nodes how to manage these workloads.

- Ansible Automation Platform Operator - For pre configuration of network, storage, bare metal resources, DNS, certificates, preparation of VMware clusters and day 2 operations the Ansible Automation platform has validated content that can help customers perform all of these tasks and more to deliver an end to end automated experience.

Additional features

Compute

OpenShift Virtualization introduces support for virtual machines as a native capability to OpenShift, Red Hat's enterprise application platform offering based on Kubernetes. The concept of using virtual machines in a "Kubernetes way" may seem strange to those that are used to traditional hypervisors. However, they will find that many of the same features and capabilities that are expected of any enterprise class virtualization solution, are available in OpenShift Virtualization.

GPU/vGPU access

If your virtual machines need to utilize (v)GPU resources, ensure that you follow the documentation for enabling access to mediated devices and enabling other features (IOMMU) as needed.

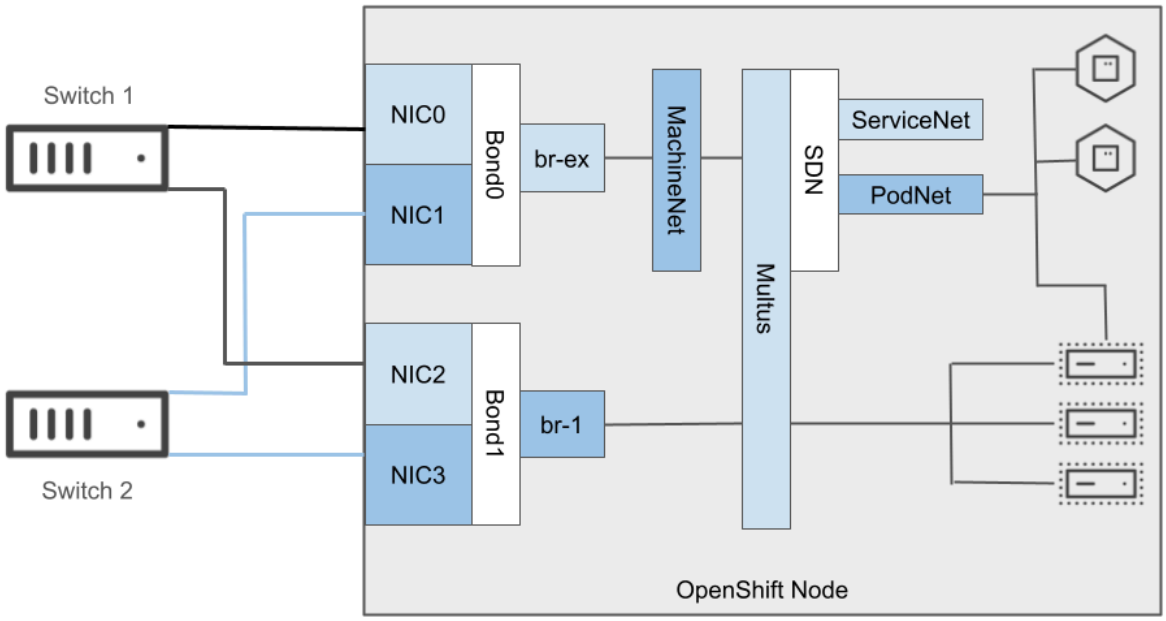
Hardware/PCI Passthrough

The PCI passthrough feature allows you to present direct access to hardware devices of host machines to the virtual guests themselves. This can be useful for a number of reasons primarily related to resource isolation for the virtual machine and the workload it is hosting. Examples include virtual machines running AI/ML based workloads, which can be granted direct access to GPU devices on the host, or perhaps a guest that requires a large amount of network bandwidth can be attached directly to a physical network adapter instead of sharing the SDN with other virtual machines. For more information related to device passthrough in OpenShift Virtualization, see the documentation here.

Network

An OpenShift cluster is configured using an overlay software-defined network (SDN) for both the Pod and Service networks. By default, VMs are configured with connectivity to the SDN and have the same features/connectivity as Pod-based applications. This means they have an internal-only IP address for being accessed across the SDN, with internal DNS resolution provided by creating a Service to abstract access to one or more VMs. On the SDN, the virtual machine can then have a specific service or port exposed through a Route like Pods. The Multus Container Network Interface (CNI), which is deployed and used with all OpenShift clusters, enables attaching multiple network interfaces to Pods and VMs, including simultaneous connections to the SDN and one-or-more external (e.g. VLAN) networks.

Host-level networking configurations are created and applied using the NMstate operator. This includes the ability to report the current configuration options, such as bonds, bridges, and VLAN tags to help segregate networking resources, as well as apply desired-state configuration for those entities.



Software-Defined Networking (SDN)

As of OpenShift 4.15, the Red Hat-provided SDN option with OpenShift is OVN-Kubernetes. OVN-Kubernetes is based on Open Virtual Network (OVN) and provides an overlay-based networking implementation. A cluster that uses the OVN-Kubernetes plugin also runs Open

vSwitch (OVS) on each node. OVN configures OVS on each node to implement the declared network configuration.

Additional SDN Options

There are many partners, such as Tigera, Cisco, Citrix, and F5, which also have SDN and other network plugins which work with OpenShift Virtualization to bring additional features and capabilities to virtual machines. Confirm with your vendor the capabilities which work with VMs.

Networking Architecture

By default, OpenShift Virtualization will conduct most communication using the SDN. There are two methods primarily for virtual machines or the applications currently running on the cluster to be accessed publicly. Like a Pod-based application, a Service can be created, mapped to an open port on the virtual machine, and a Route is used to expose external access to the application. However, this only works for HTTP(s)-based applications using port 80 and/or 443.

For other protocols and ports, using a Service with type LoadBalancer, such as MetalLB or one from partners such as F5 and Citrix, enables the application to be publicly accessible without the VM being connected directly to an L2 (VLAN) network. Deploying and configuring a load balancer provider is a post-deployment configuration option. The recommended option for this reference architecture is to use MetalLB with L3 mode if possible, or L2 mode as a fall back.

Furthermore, it is also possible to configure additional network resources to provide access to an external network so that the VM is configured with an IP address which allows the guest operating system to be accessed directly using a remote connectivity protocol like SSH. Refer to the section above for configuring additional network segments for how to set up this scenario within OpenShift Virtualization. Ansible Automation Platform provides the ability to configure additional settings and scenarios within the guest operating system as well.

For more information about network configuration options in OpenShift, please see the documentation available here.

Storage

OpenShift Virtualization uses the Persistent Volume (PV) paradigm from Kubernetes to provide storage for virtual machines. More specifically, each VM disk is stored in a dedicated persistent volume provisioned and managed by a CSI (Container Storage Interface) driver provided by Red Hat or a supported storage vendor.

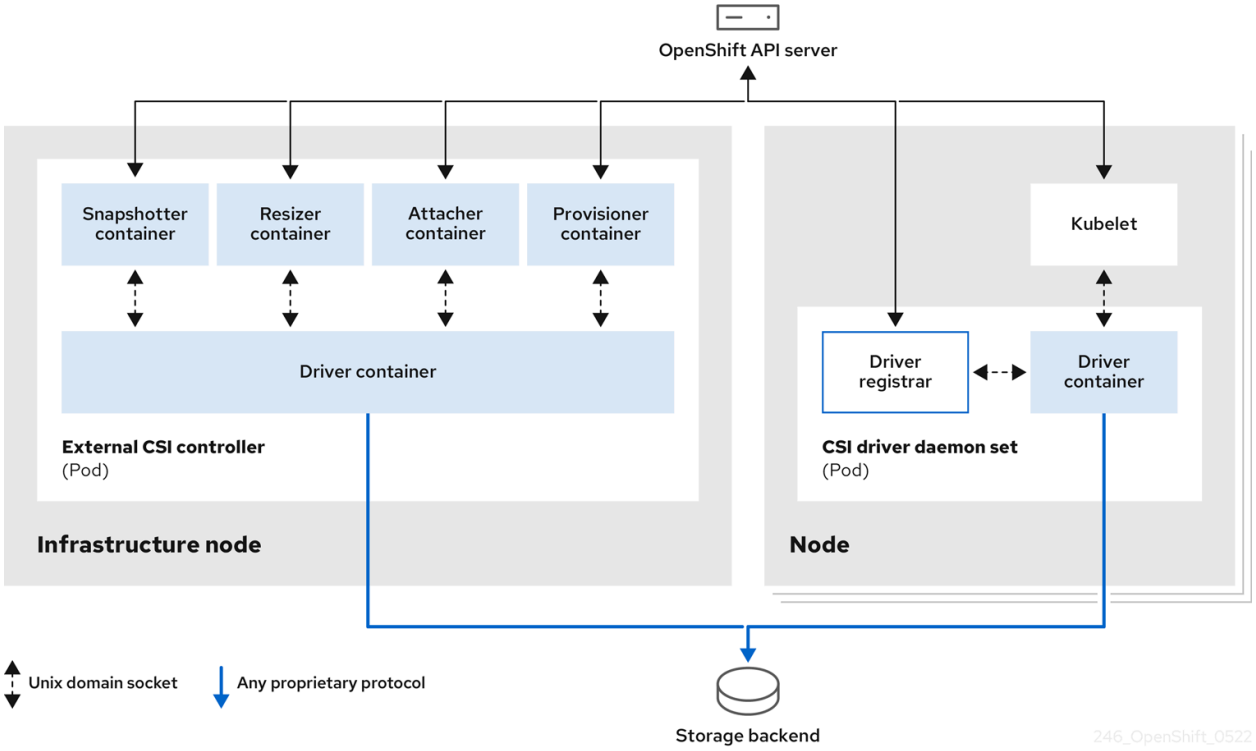
CSI Provider

This is different from the paradigm used by other virtualization offerings where a single storage device, e.g. an iSCSI/FC LUN or NFS export, is used to store many virtual machine disks. The granular provisioning, application of features, control/management, and consumption of VM disks enables the storage provider to customize and tailor the capabilities of the VM disk specifically to the workload, which is abstracted via the storage class within OpenShift.

Live migration of virtual machines requires that all virtual machine disks for the VM being migrated use PVCs with the read-write-many (RWX) access mode. This is so that the disk can be mounted to both the source and destination hypervisor node during the migration. While both block (iSCSI, Fiber Channel, etc.) and file (NFS) protocols support RWX access, you must verify with your storage vendor any additional requirements or limitations specific to the storage device being used to host the virtual machine.

The Container Storage Interface (CSI) is a standardized abstraction for managing and consuming storage within Kubernetes. OpenShift Virtualization using this paradigm means that storage from any vendor with a CSI driver that meets the needs and requirements can be used for virtual machine disks.

The diagram below provides a high-level overview of the CSI components in an OpenShift cluster. The CSI driver coordinates the creation and management of the volume on the storage device, mounting of the volume to the node hosting the virtual machine, and implementing features like snapshots and volume resizing at the storage volume level.



Multiple CSI drivers can be deployed to an OpenShift cluster to enable support of many different storage backends. This allows VMs to use various disk types, storage protocols, and access modes from different vendors to best suit the workload.

The following table describes the CSI drivers that are installed by default in an OpenShift cluster, dependent on the infrastructure platform used, along with which CSI features they support, such as volume snapshots, cloning, and resize.

It's also important to note here that not all of these storage types are available with every OpenShift cluster deployment, nor are they all currently compatible with OpenShift Virtualization. For example, with the 4.15 release, support for OpenShift Virtualization deployments requires the use of physical servers deployed on-premises, or bare metal nodes deployed in ROSA or AWS self-managed deployments. Other cloud-based deployments and platform types are in exploratory stages scheduled for a later release. This means that as of today, the storage drivers from those cloud providers cannot be used with OpenShift Virtualization, but only traditional container workloads in OpenShift.

In addition to those CSI drivers included with OpenShift by default there are a number of additional CSI drivers from partners such as Dell/EMC, Fujitsu, Hitatchi, HPE, IBM Storage

Fusion, NetApp, Portworx and more available via OperatorHub within OpenShift or from the storage vendor directly that support OpenShift Virtualization workloads. Check with your storage vendor for details on their driver's availability and capabilities. While each of these options provide basic storage functionality, they are often designed to leverage advanced features of the partner storage platform that can be used with OpenShift Virtualization.

A few examples of advanced features provided by vendor CSI drivers are listed below:

- Storage quotas - Resource quotas can be set for storage per project or across projects to ensure control over storage resources. Using quotas and limit ranges, cluster administrators can set constraints to limit the number of objects or the amount of storage resources used in a project. This helps cluster administrators better manage and allocate resources across all projects, ensuring that no project uses more than is appropriate for the cluster size. For more information about storage quotas, please see the documentation here.

- Storage multipathing - OpenShift Virtualization leverages the capabilities of the underlying operating system, Red Hat Enterprise Linux CoreOS, to provide these capabilities. Multipathing is not enabled as a default feature in CoreOS, in order to streamline the base deployed image. To enable this feature, it is recommended to use a Machine Config Operator to enable multipathing for optimal storage access to choose the best path. For more information about day 2 operations, such as enabling multipath in OpenShift, please see the documentation here.

- Multi-host PVC mounting - ODF, and other CSI (see above) options, allow for disk sharing in RWX (read/write many) access mode and with Ceph based storage clustering of disks. RWX PVCs are required for virtual machine live migration.

For a matrix of features that are provided by OpenShift Virtualization, please see the following chart:

	Virtual machine live migration	Host-assisted virtual machine disk cloning	Storage-assisted virtual machine disk cloning	Virtual machine snapshots
OpenShift Data Foundation: RBD block-mode volumes	Yes	Yes	Yes	Yes
OpenShift Virtualization: LVM Storage Operator	No	Yes	No	No
Other multi-node writable storage	Yes[1]	Yes	Yes[2]	Yes[2]
Other single-node writable storage	No	Yes	Yes[2]	Yes[2]

1. PVCs must request a `ReadWriteMany` access mode.
2. Storage provider must support both Kubernetes and CSI snapshot APIs

Red Hat

Important concepts and additional knowledge

Business Continuity

It's important in many enterprise cases to ensure that workloads deployed are highly available, fault-tolerant, and able to be recovered in the event of a disaster. High Availability in OpenShift Virtualization is provided by OpenShift and is built into the scheduler for pods, which will be explored more in depth momentarily. Fault tolerance can be addressed in a number of ways, from ensuring that enough physical resources exist to support the existing workloads should there be an infrastructure failure, and that the nodes in a cluster are organized in a manner to help insulate it from a single failure cascading across the entire cluster. Disaster recovery in OpenShift is another important topic. The ability to successfully backup an application and restore it to a remote cluster in a functional manner is required for many business applications. This is achieved using Red Hat native tools or tools available from one of our many technological partners.

High Availability

At a high level, high availability (HA) here refers to platform level attempts to ensure an application is available regardless of underlying failures of the physical infrastructure. An example scenario where HA comes into play is when a node fails, for whatever reason, and Kubernetes then automatically reschedules any lost pods onto the surviving nodes in the cluster.

VM failure - This refers to instances where individual VMs may crash or otherwise fail to perform their task. This can occur for many reasons, such as a misconfiguration; or during a node drain; or the out-of-memory process terminates the process as a result of resource contention. This can have an adverse effect on the application if appropriate resiliency and other measures are not used. Some of the steps to ensure high availability of applications in the event of VM failure and termination are documented in the OpenShift Virtualization virtual machine high availability guide.

Additionally, whether a virtual machine is restarted after failure can be controlled on a per-VM basis by setting the VM's run strategy. The default is `Always`, which will ensure that the VM is always running, unless it's shutdown via OpenShift. This means that if the VM is shutdown from

within the guest operating system, it would result in the VM being powered back on. To have the VM stay powered off after being shutdown from within the guest operating system, use the `RerunOnFailure` policy. Note that either of these will result in the VM being restarted after a node failure event if it was previously running.

For additional information, please see this documentation.

Node failure - Refers to any time a node unexpectedly becomes unavailable to the cluster. This could be because of a physical hardware failure, or an environmental issue such as a power failure. In either case, the node becomes inaccessible on the network and workloads must be rescheduled for other nodes that are available. In this case, some potential ways to mitigate the failure are:

- Have health checks in place for the nodes. These will detect when a node has failed in order to trigger the next phase of recovery: fencing.
- Understand that, by default, it takes five minutes for Kubernetes to identify and reschedule workloads from an unreachable node. For worker nodes that have been provisioned using an IPI-based install, as is done with this architecture, you may reduce this time by having node health checks in place, which will result in the node being fenced. If no IPMI/BMC connectivity is available, use the Self Node Remediation Operator to detect node failure faster.
- Fencing refers to the process of ensuring that the lost node is no longer running any workload. Within this reference architecture, the Fencing Agents Remediation method is used to power cycle the physical server using IPMI/BMC integration.

Cluster failure - Cluster high availability depends on the control plane staying healthy in the event of failure. If the control plane fails, or otherwise becomes inaccessible, new workloads will be unable to schedule and the compute nodes may begin to shutdown VMs after a timeout. The scheduling of Pods is one of the most critical functions provided by the control plane. With OpenShift 4, all clusters (with the exception of single node OpenShift) have three control plane nodes, each one also hosting an etcd member. With three nodes, the control plane will continue to function even with one failed node. If two nodes fail, then the control plane cannot process cluster changes, and becomes unavailable until a quorum is restored. If all three OpenShift control plane nodes happen to fail at the same time then the cluster is offline and unreachable until the cluster can be restored to working order.

Disaster Recovery

In general, if separate availability zones cannot be guaranteed for the cluster control plane and worker nodes to assist with mitigation of a site failure, then the best option available is to deploy additional OpenShift clusters across multiple sites and establish a disaster recovery policy. This policy should define which actions need to be taken to recover applications should there be a cluster failing disaster at one site. In some cases this can be achieved by placing a global load balancer in front of a number of clusters at remote sites, ensuring that one cluster can still service applications should its partner fail. However, this does require some level of application awareness and monitoring to ensure that the correct instance is active and any persistent data used by the application is replicated appropriately.

To achieve this, Red Hat has made available the OpenShift APIs for Data Protection (OADP) as an operator in OpenShift to enable users to create backups and restore workload in OpenShift. For more details on OADP, please see the documentation available here.

The specific storage product used for OpenShift Virtualization may offer the ability to replicate or otherwise protect the virtual machine disks and metadata. The disaster recovery guide offers several options for how to protect and recover virtual machines with varying recovery time object (RTO) and recovery point object (RPO) goals.

Additional options for backup, restore, and disaster recovery exist by making use of products from a number of Red Hat Partners which provide the ability to back up and restore stateful applications from one OpenShift cluster to another. Several of these partners focus on backup and recovery for applications in OpenShift and also currently support OpenShift Virtualization workloads. In addition to these partners that focus exclusively on backup and recovery operations, there are storage vendors with native backup functionality that also support OpenShift Virtualization.

This is a brief listing of some partners that provide the functionality to backup and restore virtual machines as well as containerized workloads in Red Hat OpenShift :

- IBM Storage Fusion
- Portworx PX-Backup and Metro-DR
- OpenShift Data Foundation Metro-DR and Regional-DR
- Storware Backup and Recovery
- Trilio for Kubernetes
- VEEAM Kasten

*Please note that this list is not intended to be exhaustive,a more complete list is available at https://red.ht/workswithvirt, or reach out to your preferred storage or backup/recovery provider to confirm support for OpenShift Virtualization.

Scalability

The upward bounds of an OpenShift Virtualization deployment is based on the tested and supported maximums of Red Hat OpenShift itself. The table below provides the basic information for scaling of resources available to a single cluster. More information on OpenShift tested maximums can be found in the documentation here, virtualization specific maximums can be found here.

Component	Tested Maximum
Nodes in a cluster	500 [1][2]
Virtual Machines in a cluster	10,000
CPUs per Virtual Machine	216[3]
RAM per Virtual Machine	16 TB[4]
Single Disk Size per Virtual Machine	6 TB[5]

[1] This number is lower than the overall limitation of OpenShift

[2] While 500 node clusters are tested, it is suggested to run cluster sizes of less than 100 nodes

[3] Theoretical limit is 710 on x86_64 infrastructure, due to underlying constraints of KVM and RHCOS

[4] Limited based on underlying constraints of KVM and RHCOS

[5] This is the tested size, the supported maximum is the same as RHEL.

Subscription Information

OpenShift Virtualization is included in all OpenShift subscriptions. Subscription models for bare metal are purchased based on the socket count in a physical node, and a single subscription is allocated to 1-2 sockets, with up to 64 cores total. A bare metal system running OpenShift Virtualization then has the ability to run an unlimited number of virtualized Red Hat Enterprise Linux (RHEL) guest operating system instances included as a part of that subscription. While this subscription covers RHEL guests, it does not cover virtualized OpenShift clusters hosted by the physical OpenShift Virtualization cluster. Virtual OpenShift clusters deployed

non-integrated or using the hosted control plane feature are entitled using core-based subscriptions, in the same manner as OpenShift deployed on any hypervisor platform.

	Socket Subscription Required	Core Subscription Required
OpenShift bare metal	Up to 64 physical cores, 1 or 2 sockets.*	N/A
Virtual OpenShift on OpenShift bare metal	N/A	Virtual OpenShift entitled by compute node vCPU.**
Mixed virtual machines on OpenShift bare metal	Up to 64 physical cores, 1 or 2 sockets.*	Virtual OpenShift entitled by compute node vCPU.**

* Hardware with greater than 64 physical cores will require additional subscriptions per node.

** Maximum core subscription for virtual OpenShift capped as physical core limit of host cluster.

Solution Summary

OpenShift Virtualization is a fully developed virtualization solution utilizing the type-1 KVM hypervisor from Red Hat Enterprise Linux with the powerful features and capabilities of OpenShift for managing hypervisor nodes, virtual machines, and their consumers. By providing a single control plane for managing both containers and VMs, OpenShift can streamline many administrative actions and assist with expediting the deployment of full application stacks that involve both of these resource types. Customers should also be assured of the value provided by OpenShift Virtualization in comparison to other enterprise virtualization solutions, as it also maintains many of the same hypervisor features expected of such a solution, like live migration, virtual guest cloning, and integration with third-party tools for virtual machine management.

Additional Links and References

Documentation:

About OpenShift Virtualization:
https://docs.openshift.com/container-platform/4.16/virt/about_virt/about-virt.html

Videos:

In the Clouds Episode 33: https://www.youtube.com/watch?v=qzrTfJfbb8o
Ask an OpenShift Admin Episode 89: https://www.youtube.com/watch?v=oUm7yftZI20

Appendix A. Detailed resource calculations

CPU capacity calculation

```
(((physical_cpu_cores - odf_requirements - control_plane_requirements) *
node_count * overcommitment_ratio) * (1 -ha_reserve_percent)) * (1 -
spare_capacity_percent)
```

- physical_cpu_cores = the number of physical cores available on the node.
- odf_requirements = the amount of resources reserved for ODF. A value of 30 cores was used for the example architectures based on the *balanced* performance profile found in documentation, here.
- control_plane_requirements = the amount of CPU reserved for the control plane workload. A value of 4 cores was used for the example architectures.
- node_count = the number of nodes with this geometry. For small, all nodes were equal. For medium, the nodes are mixed-purpose, so the previous steps would need to be repeated for each node type, taking into account the appropriate node type.
- overcommitment_ratio = the amount of CPU overcommitment. A ratio of 4:1 was used for this document.

- spare_capacity = the amount of capacity reserved for spare/burst. A value of 10% was used for this document.
- ha_reserve_percent = the amount of capacity reserved for recovering workload lost in the event of node failure. For the small example, a value of 25% was used, allowing for one node to fail. For the medium example, a value of 20% was used, allowing for two nodes to fail.

Memory capacity calculation

```
((total_node_memory - odf_requirements - control_plane_requirements) *
soft_eviction_threshold_percent * node_count) * (1 - ha_reserve_percent)
```

- total_node_memory = the total physical memory installed on the node.
- odf_requirements = the amount of memory assigned to ODF. A value of 96 GiB was used for the example architectures in this document based on the *performance* performance profile found in documentation, here.
- control_plane_requirements = the amount of memory reserved for the control plane functions. A value of 24GiB was used for the example architectures.
- soft_eviction_threshold_percent = the value at which soft eviction is triggered to rebalance resource utilization on the node. Unless all nodes in the cluster exceed this value, it's expected that the node will be below this utilization. A value of 90% was used for this document.
- node_count = the number of nodes with this geometry. For small, all nodes were equal. For medium, the nodes are mixed-purpose, so the previous steps would need to be repeated for each node type, taking into account the appropriate node type.
- ha_reserve_percent = the amount of capacity reserved for recovering workload lost in the event of node failure. For the small example, a value of 25% was used, allowing for one node to fail. For the medium example, a value of 20% was used, allowing for two nodes to fail.

ODF storage capacity calculation

```
(((disk_size * disk_count) * node_count) / replica_count) * (1 -
utilization_percent)
```

- disk_size = the size of the disk(s) used. 4TB and 8TB disks were used in the example architectures.
- disk_count = the number of disks of disk_size in the node.
- node_count = the number of nodes with this geometry. For small, all nodes were equal. For medium, the nodes are mixed-purpose, so the previous steps would need to be repeated for each node type taking into account the appropriate node type.
- replica_count = the number of copies ODF stores of the data for protection/resiliency. A value of 3 was used for this document.
- utilization_percent = the desired threshold of capacity used in the ODF instance. A value of 65% was used for this document.

Appendix B. Change log

May 2024 - version 1.0.0

- Initial release.

June 2024 - version 1.0.1

- Added `allow-extra-patch-ports` to OVS configuration for virtual machine networking.

September 2024 - version 1.0.2

- Addressed outstanding comments.
- Updated values for information that has changed in 4.16.
- Added additional DR and storage partners into appropriate sections.