Red Hat
OpenShift

OpenShift Virtualization

Technical Workshop for FSI

Mike Pagan
Principal Platform SA
Red Hat

Agenda

- Review of Openshift virtualization (kubevirt)
- Management of Virtual Machines
    - Creation, Modification, and Retirement of VMs
    - Importing Virtual Machines
    - Viewing Virtual Machine Details
    - Virtual Machine Metrics
- Deep Dive on Openshift virtualization Technologies
- Openshift virtualization Cluster Architecture Options
- Deep Dive on VM Resources
    - Compute
    - Storage
    - Netrork
- Comparison of Openshift virtualization with Traditional Virtualization

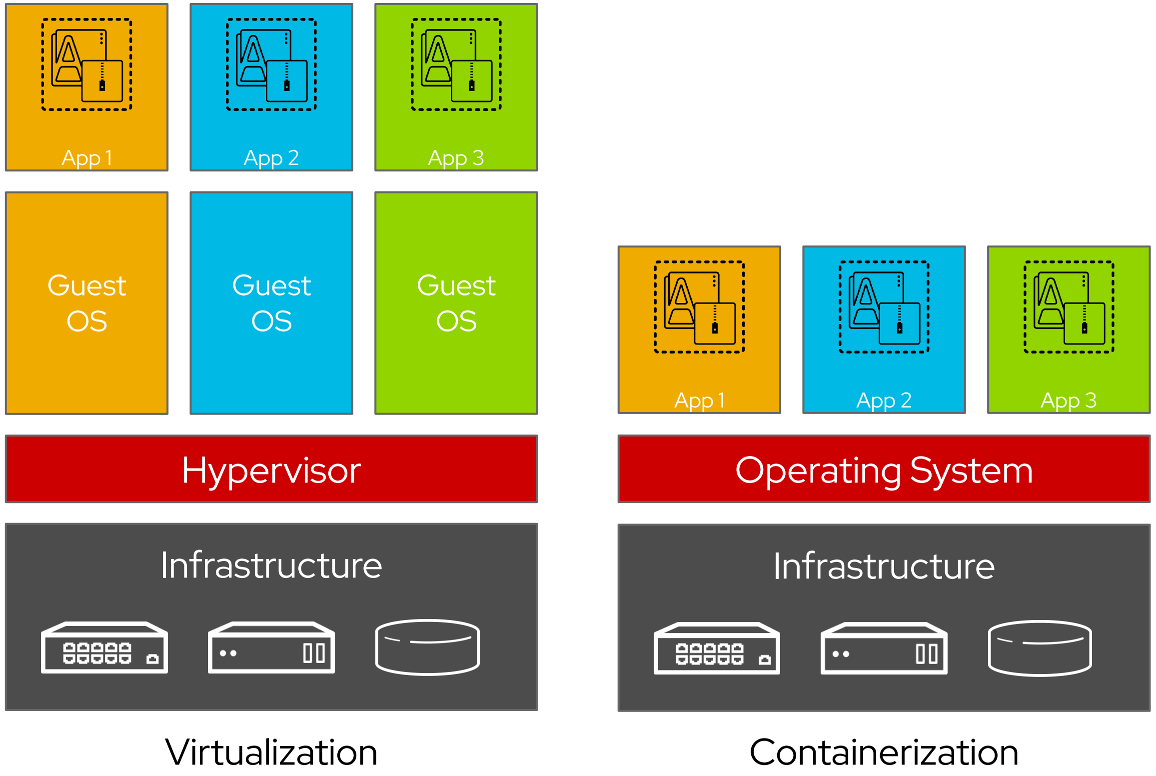
But first: Some Introductions

- Introductions
  - Who Am I
  - My history with Opeshift virtualization
- Logistics
  - Webex minutia
  - Time management
- Audience
  - Engineers & Admins

What is OpenShift Virtualization?

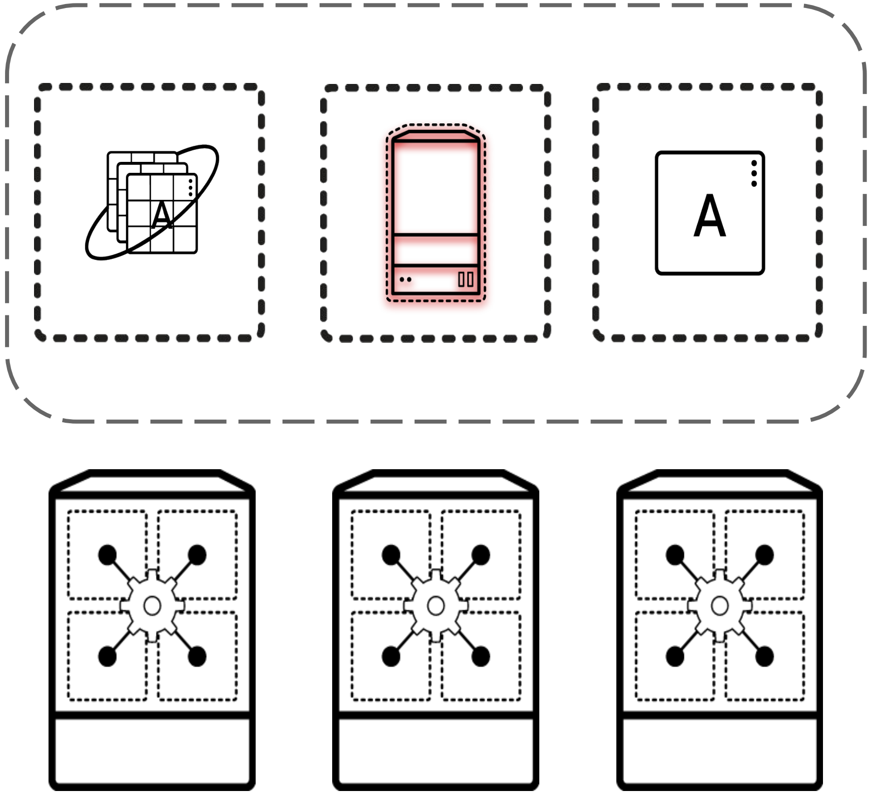
Containers are not virtual machines

- Containers are process isolation
- Kernel namespaces provide isolation and cgroups provide resource controls
- No hypervisor needed for containers
- Contain only binaries, libraries, and tools which are needed by the application
- Ephemeral



Virtual machines can be put into containers

- A KVM virtual machine is a process
- Containers encapsulate processes
- Both have the same underlying resource needs:
  - Compute
  - Network
  - (sometimes) Storage

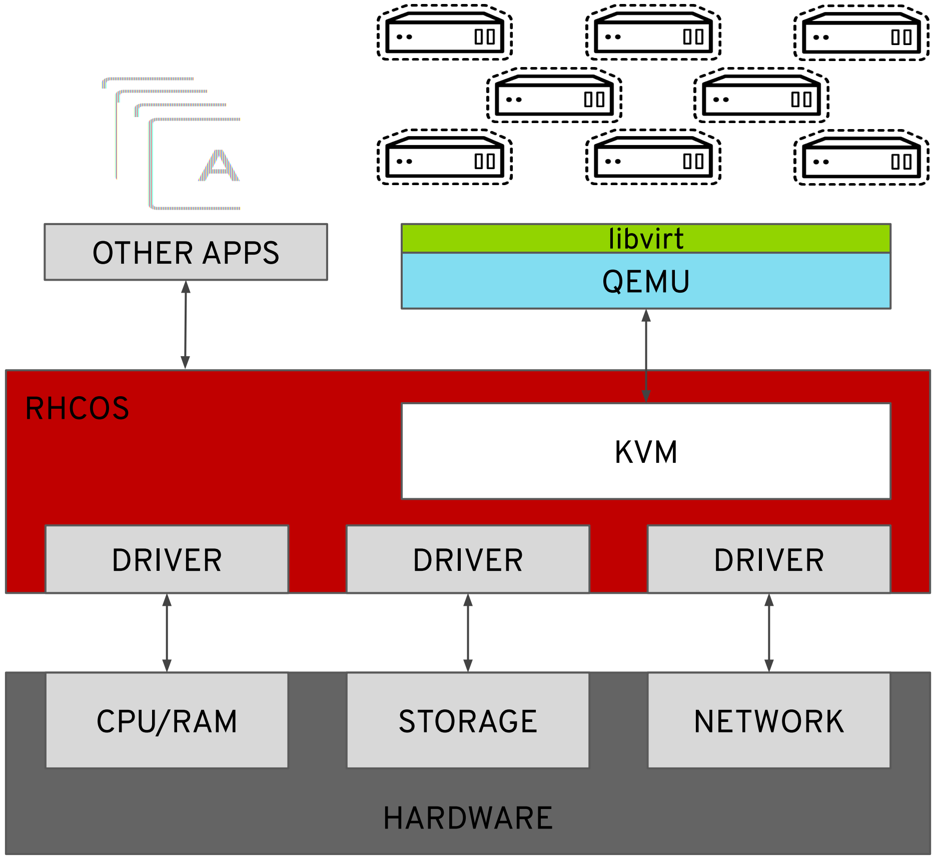


OpenShift Virtualization

- Virtual machines
  - Running in containers
  - Using the KVM hypervisor
- Scheduled, deployed, and managed by Kubernetes
- Integrated with container orchestrator resources and services
  - Traditional Pod-like SDN connectivity and/or connectivity to external VLAN and other networks via multus
  - Persistent storage paradigm (PVC, PV, StorageClass)

VM containers use KVM

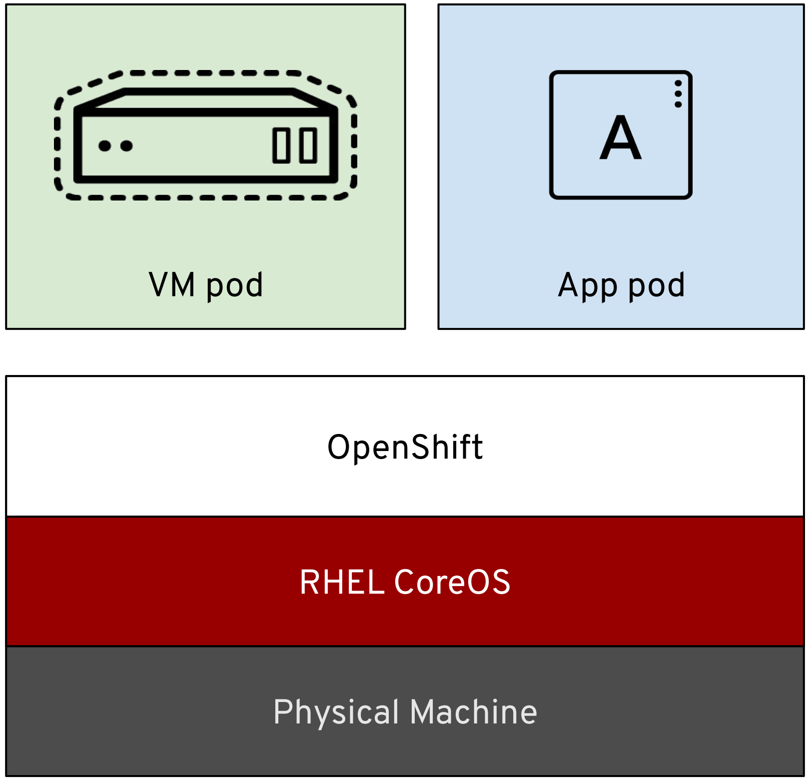
- OpenShift Virtualization uses KVM, the Linux kernel hypervisor
- KVM is a core component of the Red Hat Enterprise Linux kernel
    - KVM has 10+ years of production use: Red Hat Virtualization, Red Hat OpenStack Platform, and RHEL all leverage KVM, QEMU, and libvirt
- QEMU uses KVM to execute virtual machines
- `libvirt` provides a management abstraction layer



Built with Kubernetes

Virtual machines in a container world

- Provides a way to transition application components which can't be directly containerized into a Kubernetes system
  - Integrates directly into existing k8s clusters
  - Follows Kubernetes paradigms:
    - Container Networking Interface (CNI)
    - Container Storage Interface (CSI)
    - Custom Resource Definitions (CRD, CR)
- Schedule, connect, and consume VM resources as container-native

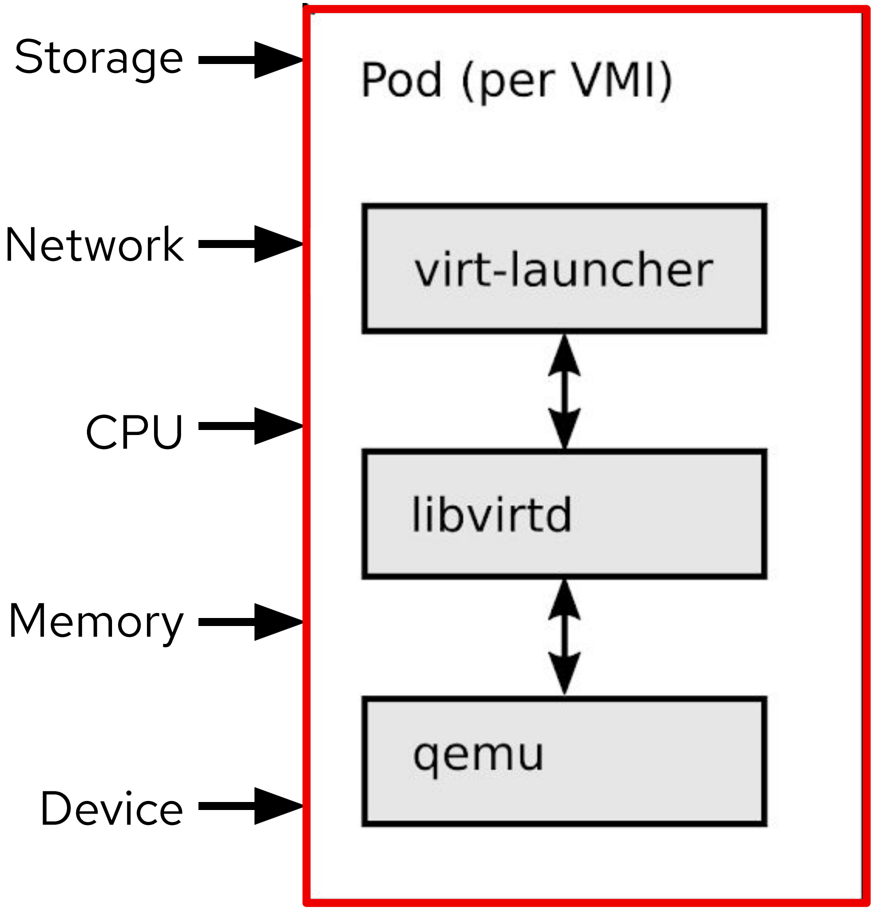


Virtualization native to Kubernetes

- Operators are a Kubernetes-native way to introduce new capabilities
- New CustomResourceDefinitions (CRDs) for native VM integration, for example:
  - `VirtualMachine`
  - `VirtualMachineInstance`
  - `VirtualMachineInstanceMigration`
  - `DataVolume`

```yaml
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    app: demo
    flavor.template.kubevirt.io/small: "true"
  name: rhel
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1alpha1
    kind: DataVolume
    metadata:
      creationTimestamp: null
      name: rhel-rootdisk
    spec:
      pvc:
        accessModes:
        - ReadWriteMany
        resources:
          requests:
            storage: 20Gi
        storageClassName: managed-nfs-storage
        volumeMode: Filesystem
```

Containerized virtual machines



Kubernetes resources

- Every VM runs in a launcher pod. The launcher process will supervise, using libvirt, and provide pod integration.

Red Hat Enterprise Linux

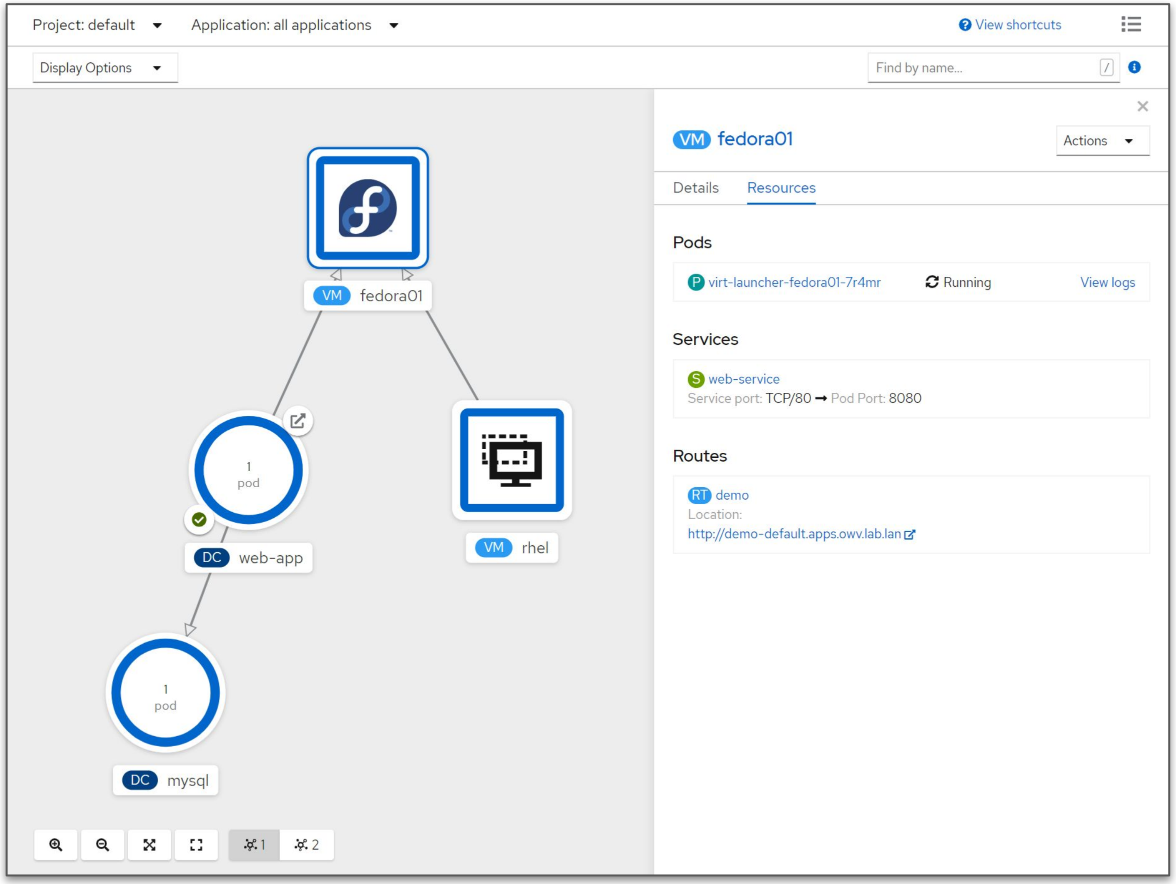
- libvirt and qemu from RHEL are mature, have high performance, provide stable abstractions, and have a minimal overhead.

Security - Defense in depth

- Immutable RHCOS by default, SELinux MCS, plus KVM isolation - inherited from the Red Hat Portfolio stack

Using VMs and containers together

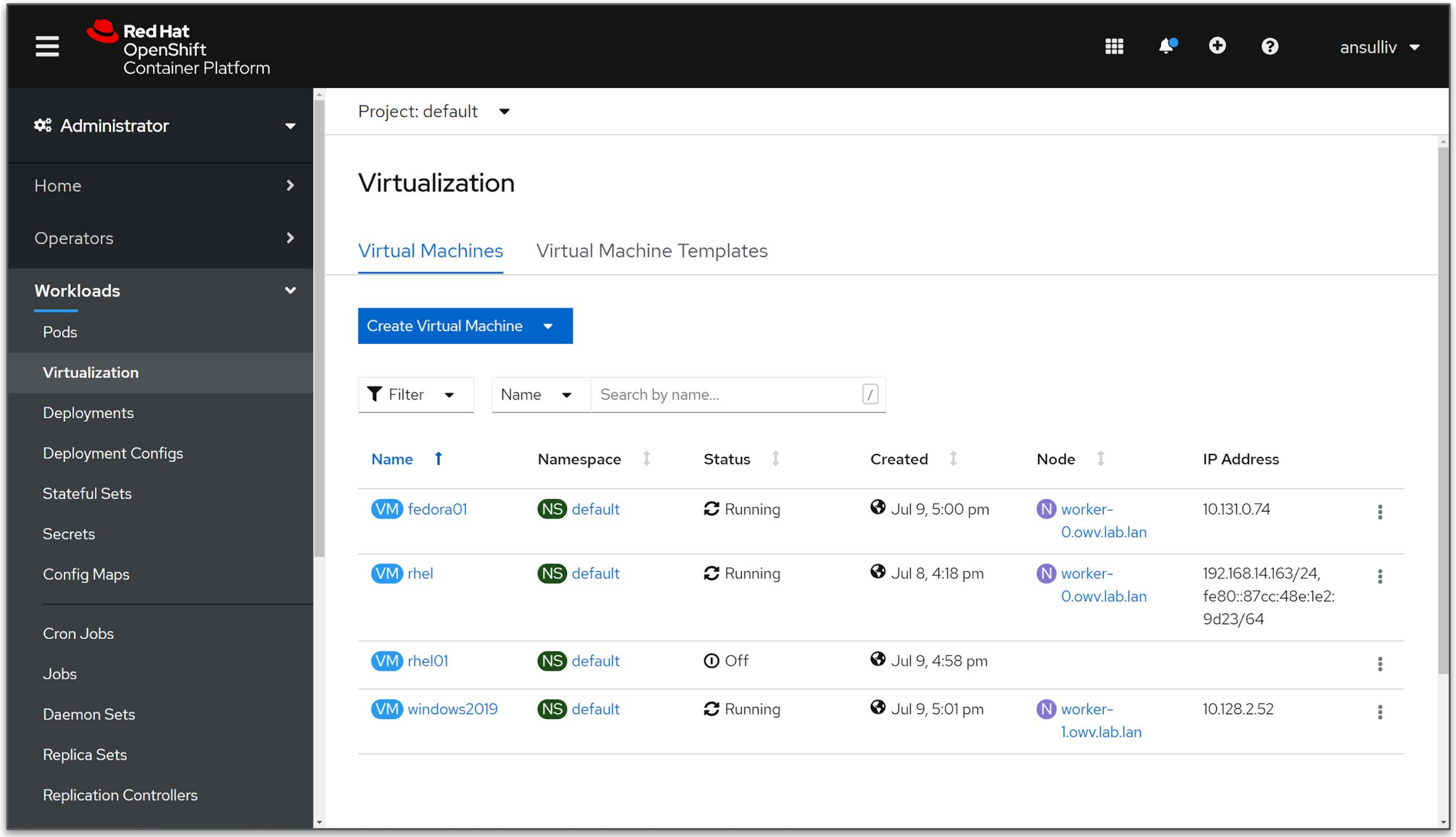
- Virtual Machines connected to pod networks are accessible using standard Kubernetes methods:
  - Service
  - Route
  - Ingress
- Network policies apply to VM pods the same as application pods
- VM-to-pod, and vice-versa, communication happens over SDN or ingress depending on network connectivity



Managin VMs with OpenShift

Virtual Machine Management

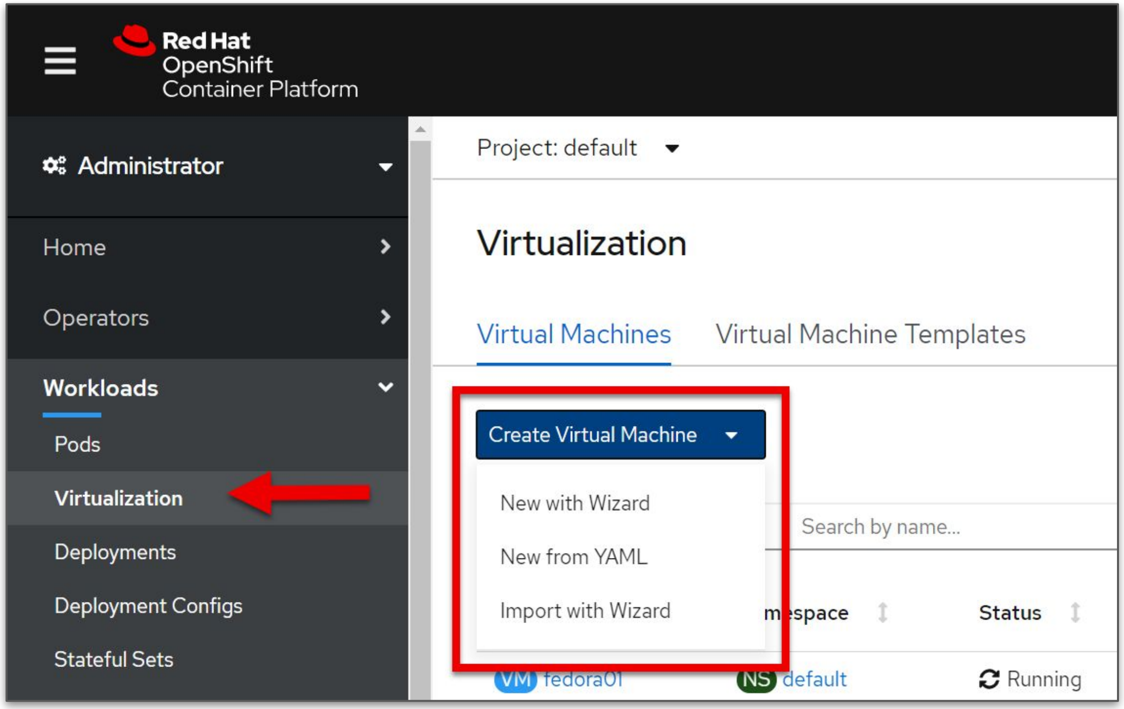
- Create, modify, and destroy virtual machines, and their resources, using the OpenShift web interface or CLI
- Use the `virtctl` command to simplify virtual machine interaction from the CLI



Create VMs

Virtual Machine creation

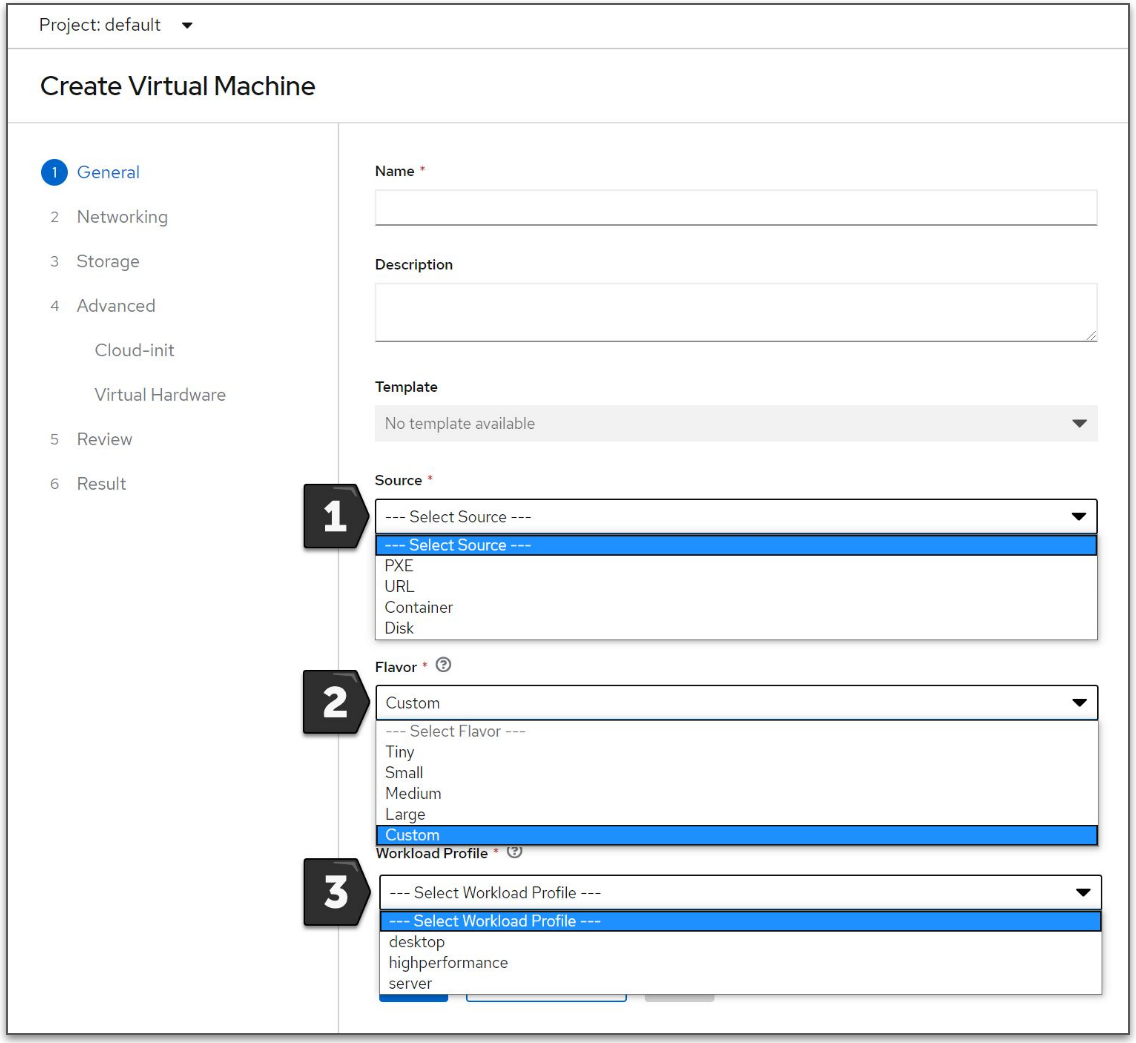
- Streamlined and simplified creation via the GUI or create VMs programmatically using YAML
- Full configuration options for compute, network, and storage resources
  - Clone VMs from templates or import disks using DataVolumes
  - Pre-defined and customizable presets for CPU/RAM allocations
  - Workload profile to tune KVM for expected behavior
- Import VMs from VMware vSphere or Red Hat Virtualization



Red Hat

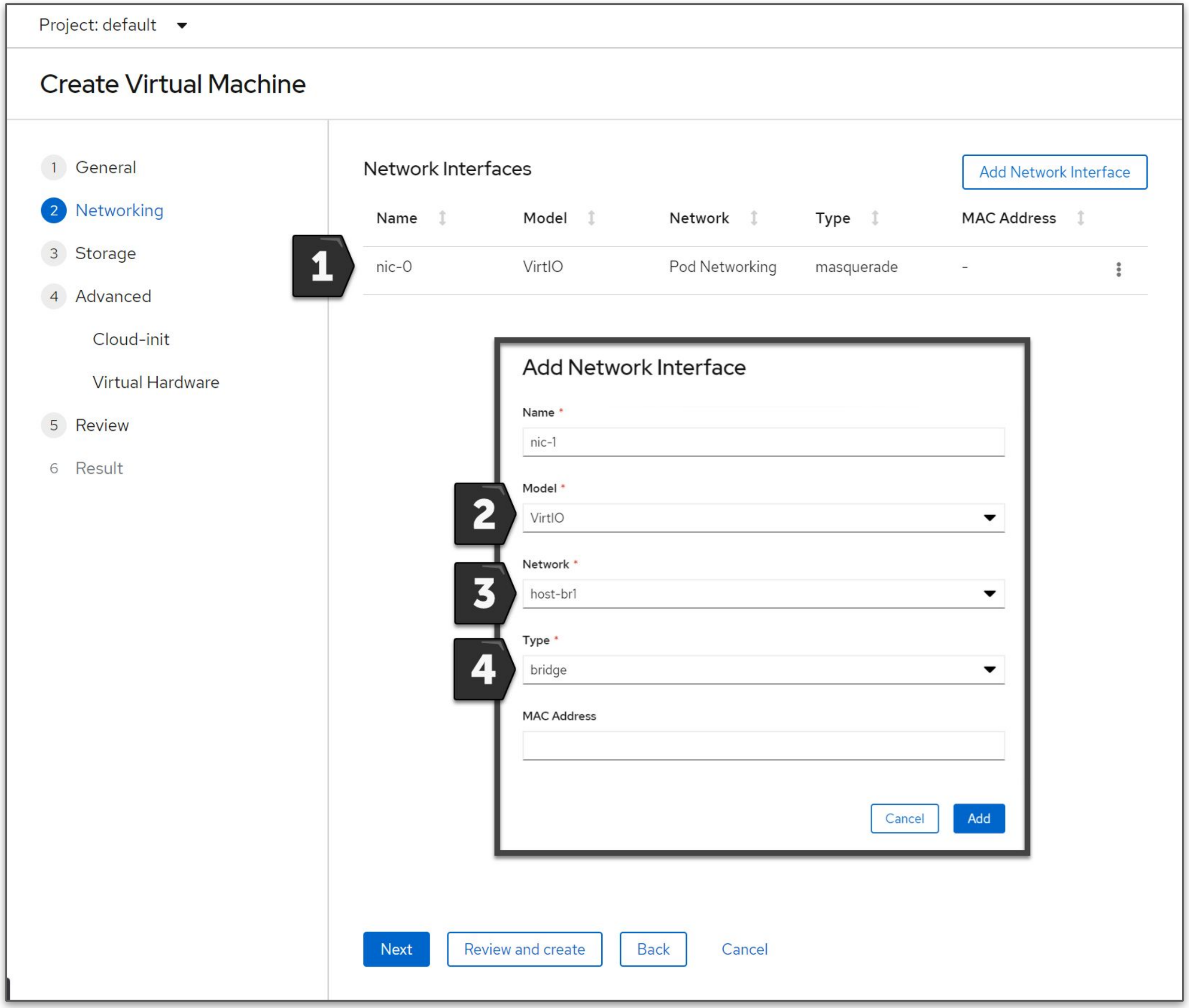
Create Virtual Machine - General

- Source represents how the VM will boot
    - Boot via PXE, optionally diskless
    - URL will import a QCOW2 or raw disk image using a DataVolume
    - Container uses a container image, pulled from a registry, for the disk
    - Disk uses an existing PVC
- Flavor represents the preconfigured CPU and RAM assignments
    - Tiny = 1 vCPU and 1GB RAM, Small = 1 vCPU and 2GB RAM, etc.
- Workload profile defines the category of workload expected and is used to set KVM performance flags



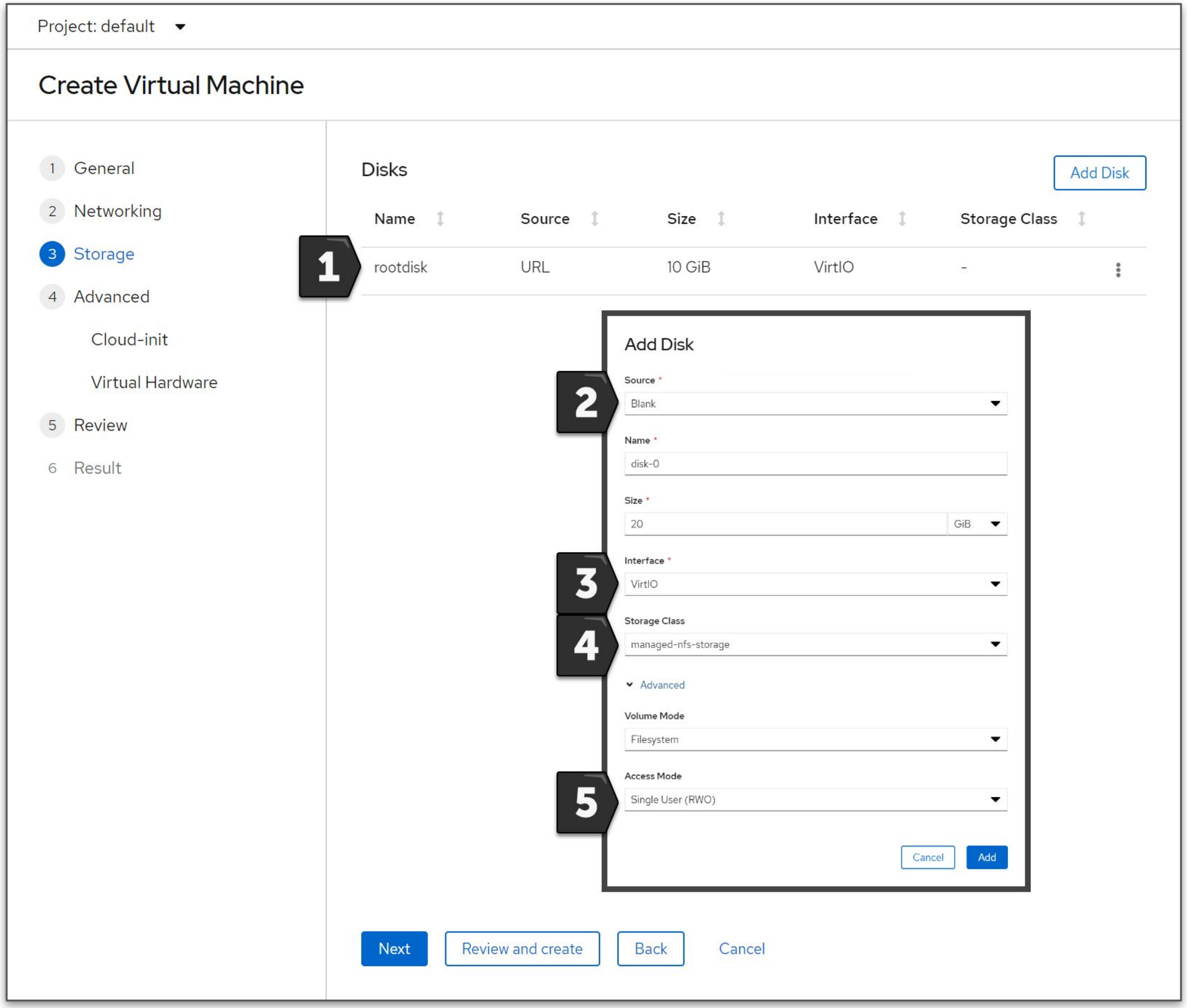
Create Virtual Machine - Networks

- Add or edit network adapters
- One or more network connections
  - Pod network for the default SDN
  - Additional multus-based interfaces for specific connectivity
- Multiple NIC models for guest OS compatibility or paravirtualized performance with VirtIO
- Masquerade, bridge, or SR-IOV connection types
- MAC address customization if desired



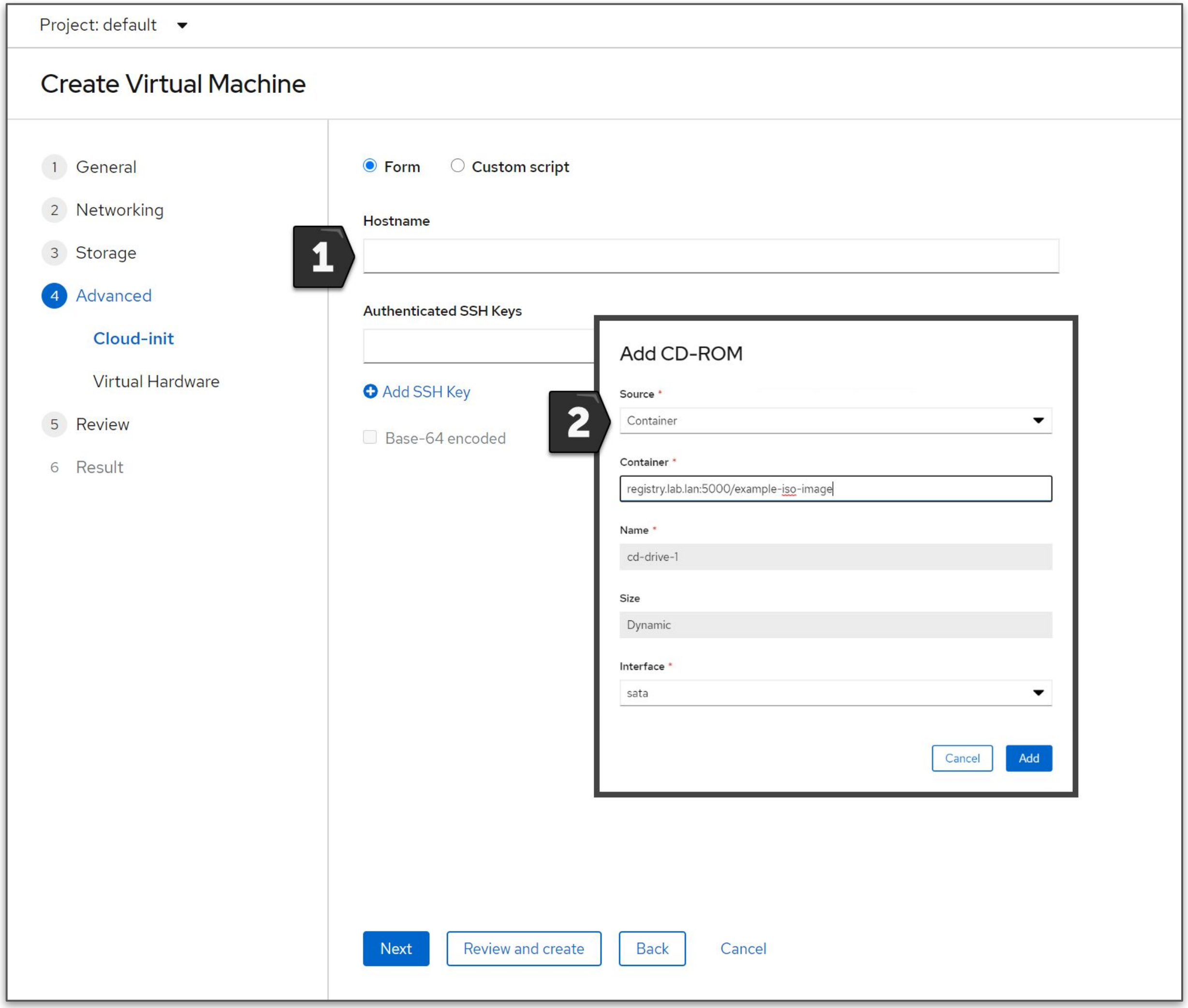
Create Virtual Machine - Storage

- Add or edit persistent storage
- Disks can be sourced from
  - Imported QCOW2 or raw images
  - New or existing PVCs
  - Clone existing PVCs
- Use SATA/SCSI interface for compatibility or VirtIO for paravirtual performance
- For new or cloned disks, select from available storage classes
  - Customize volume and access mode as needed



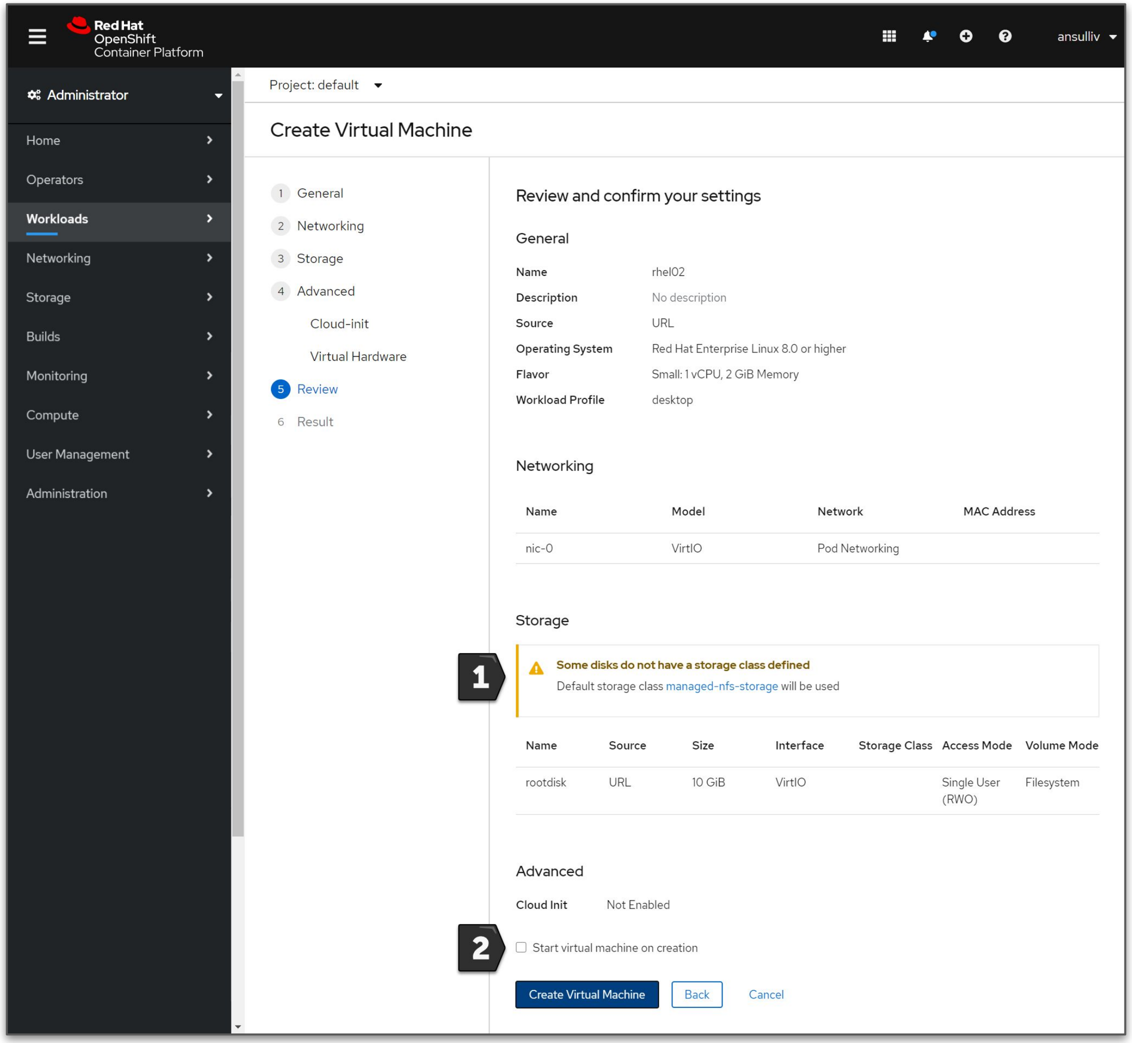
Create Virtual Machine - Advanced

- Customize the operating system deployment using cloud-init scripts
  - Guest OS must have cloud-init installed
  - RHEL, Fedora, etc. cloud images
- Attach ISOs to the VM CD/DVD drive
  - ISOs stored in container images (registry), existing PVC, or imported from URL



Create Virtual Machine - Review

- A summary of the decisions made
- Warnings and other important information about the configuration of the VM are displayed
- Choose to automatically power on the VM after creation



Import VMs

Virtual Machine Import

- Wizard supports importing from VMware or Red Hat Virtualization
  - Single-VM workflow
- VMware import uses VDDK to expedite the disk import process
  - User is responsible for downloading the VDDK from VMware and adding it to a container image
- Credentials stored as Secrets
- `ResourceMapping` CRD configures default source -> destination storage and network associations

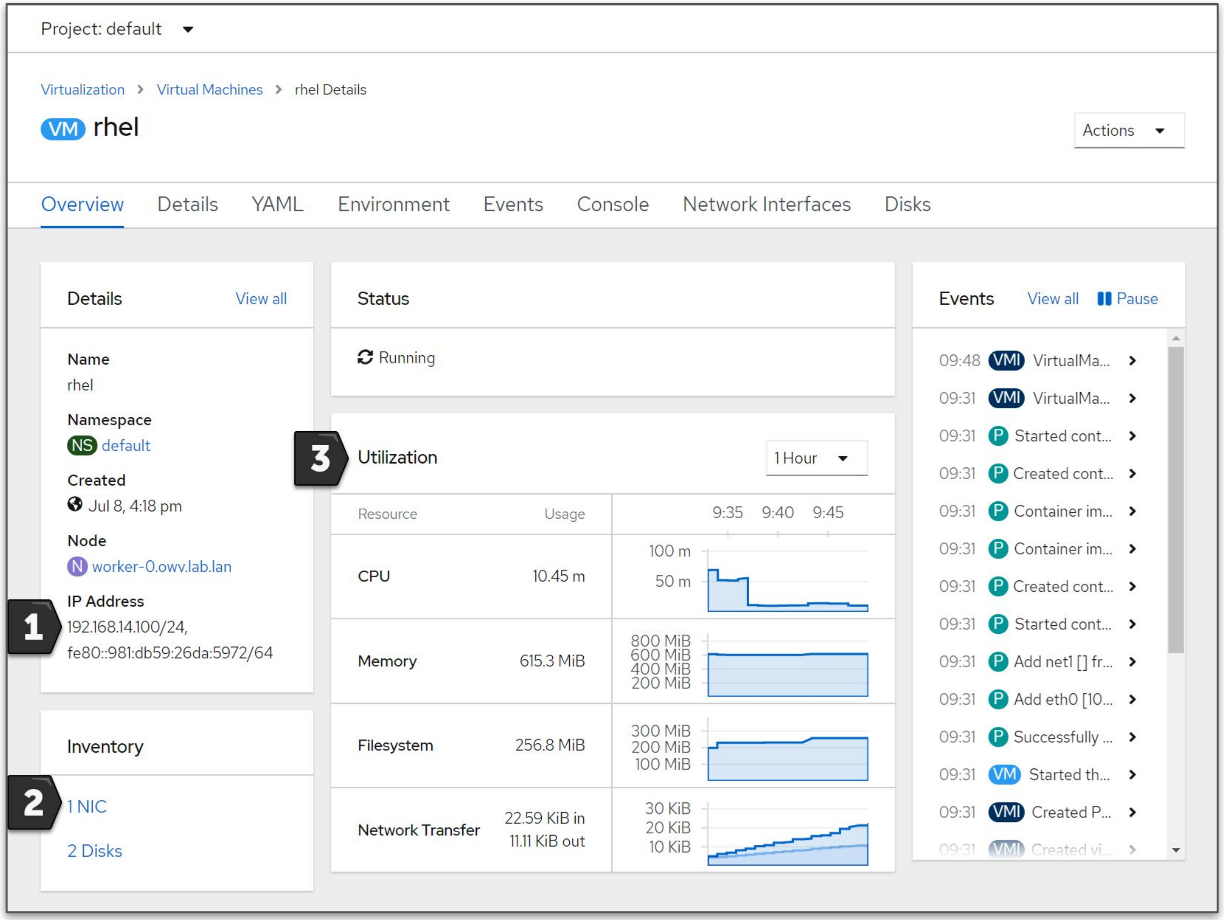
Add image here

View / manage VMs

Red Hat

Virtual Machine - Overview

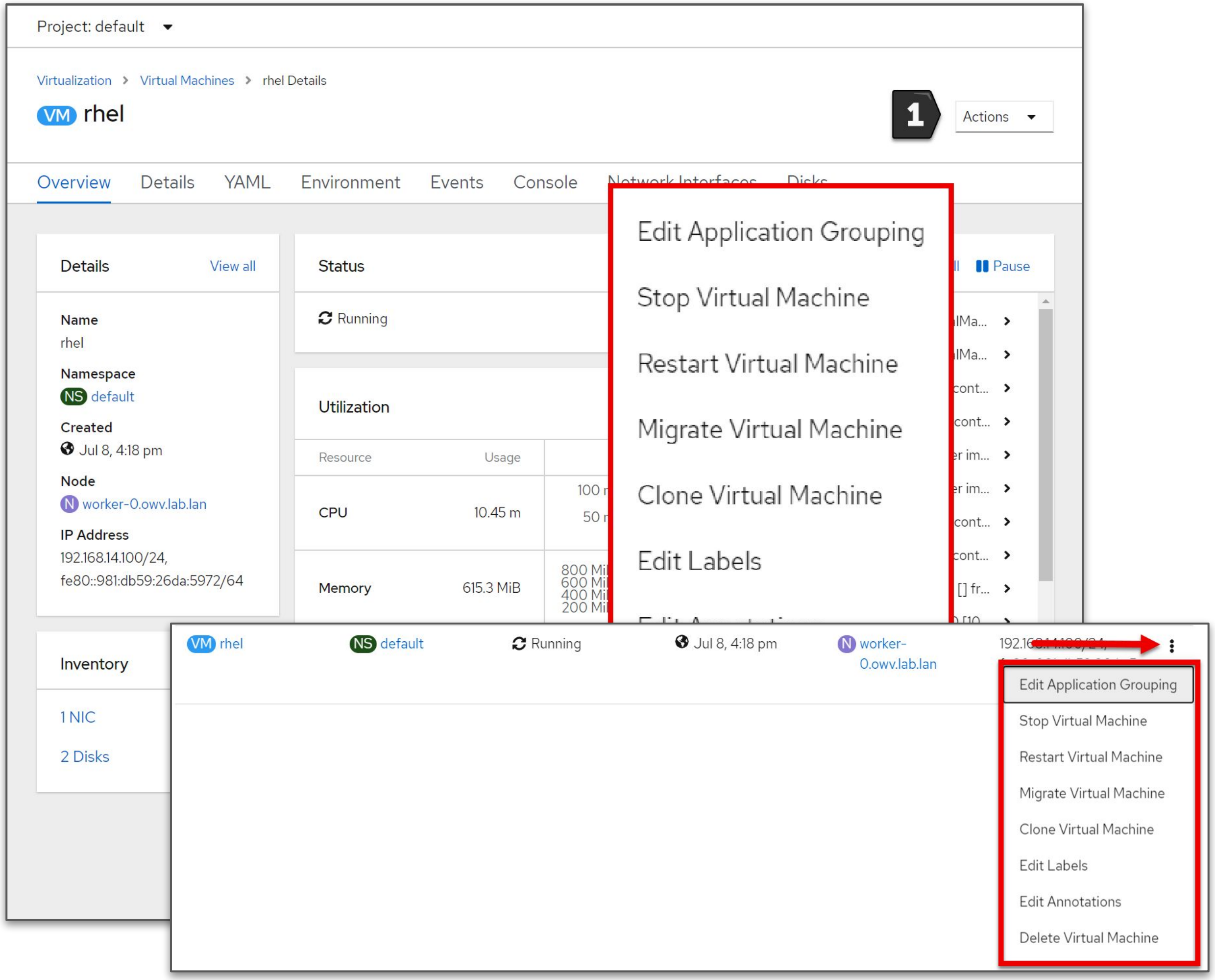
- General overview about the virtual machine
- Information populated from guest when integrations are available
  - IP address
- Inventory quickly shows configured hardware with access to view/manage
- Utilization reporting for CPU, RAM, disk, and network



V0000000

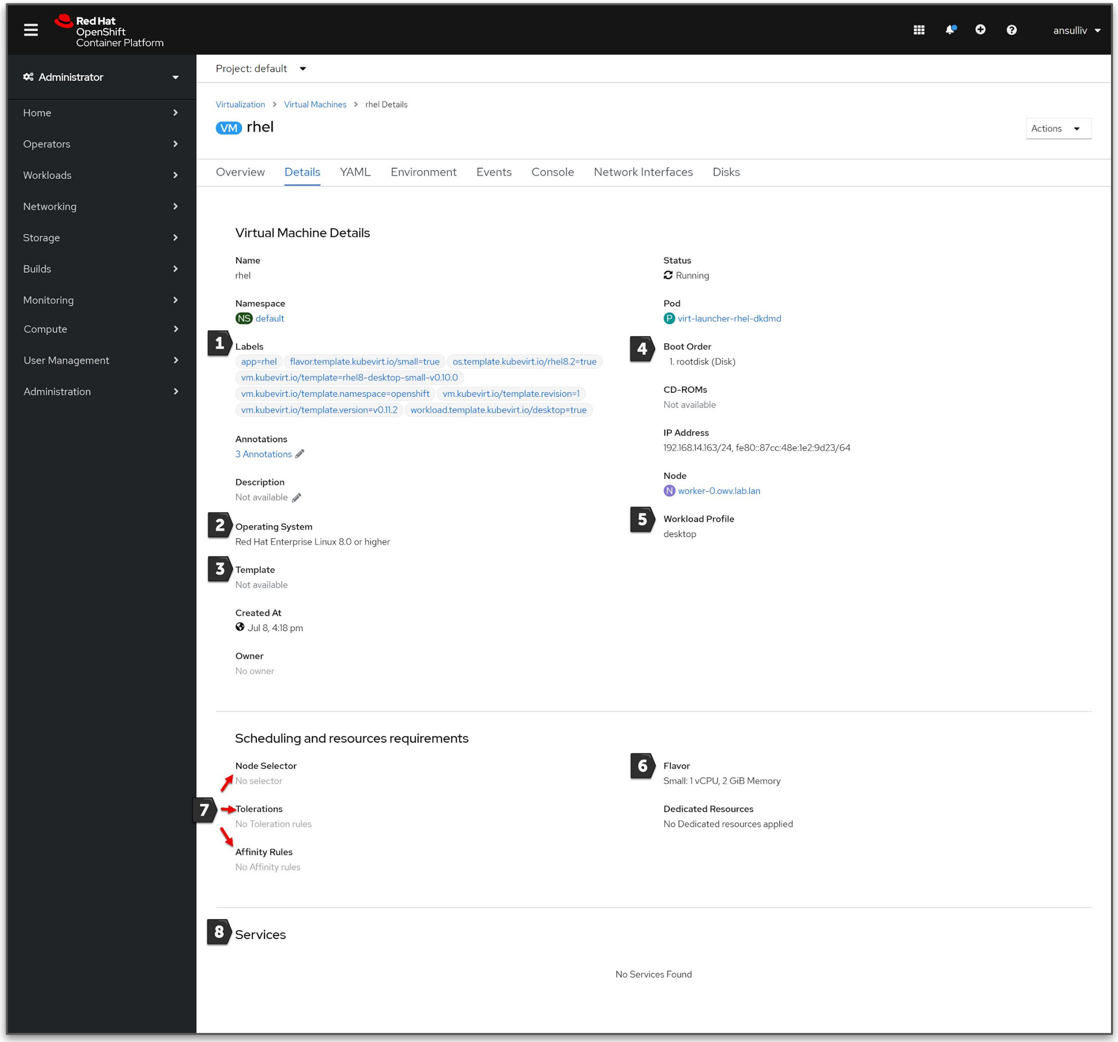
Virtual Machine - Actions

- Actions menu allows quick access to common VM tasks
  - Start/stop/restart
  - Live migration
  - Clone
  - Edit application group, labels, and annotations
  - Delete
- Accessible from all tabs of VM details screen and the VM list



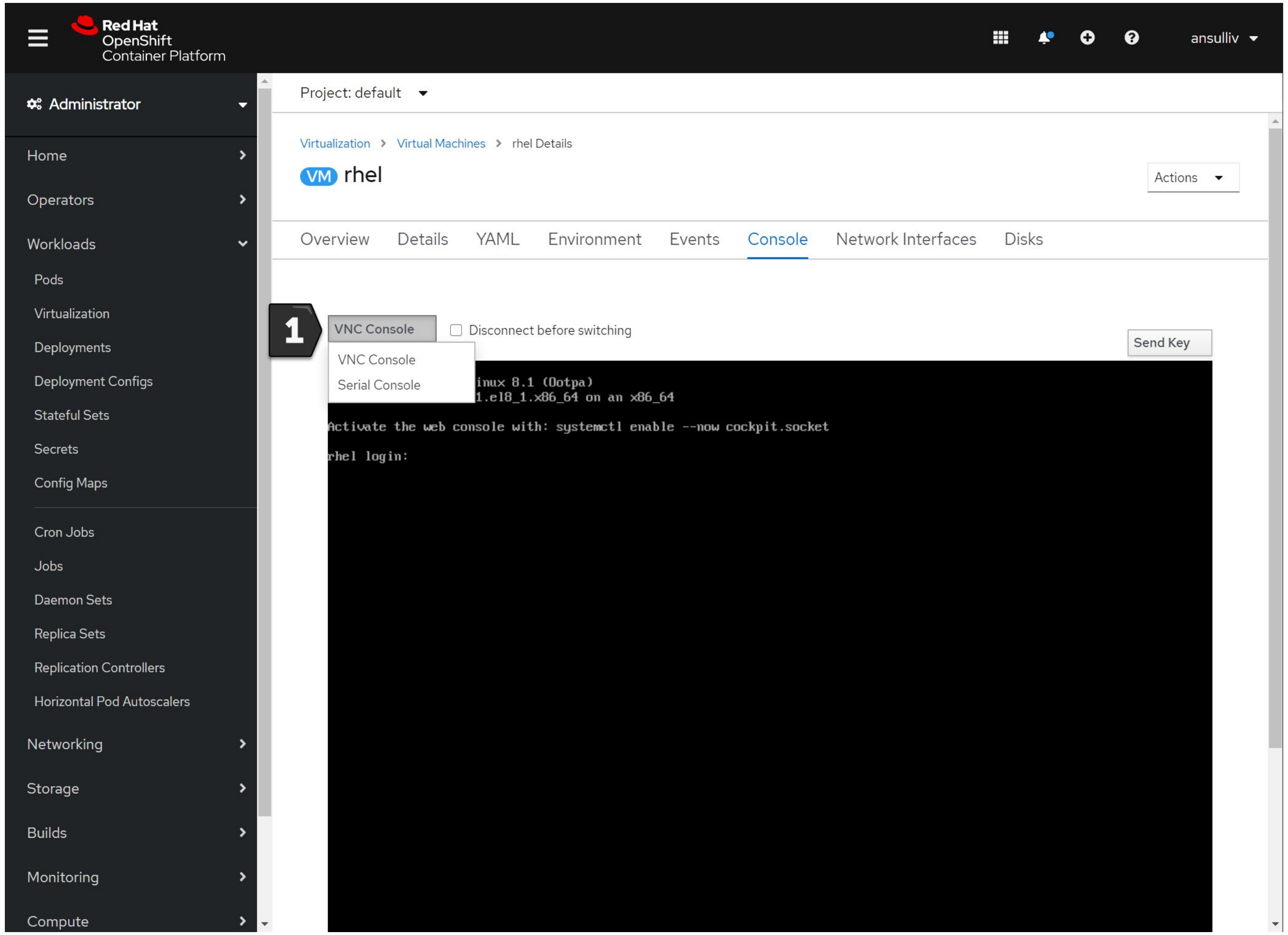
Virtual Machine - Details

- Details about the virtual machine
  - Labels, annotations
  - Configured OS
  - Template used, if any
  - Configured boot order
  - Associated workload profile
  - Flavor
- Additional details about scheduling
  - Node selector, tolerations, (anti)affinity rules
- Services configured for the VM



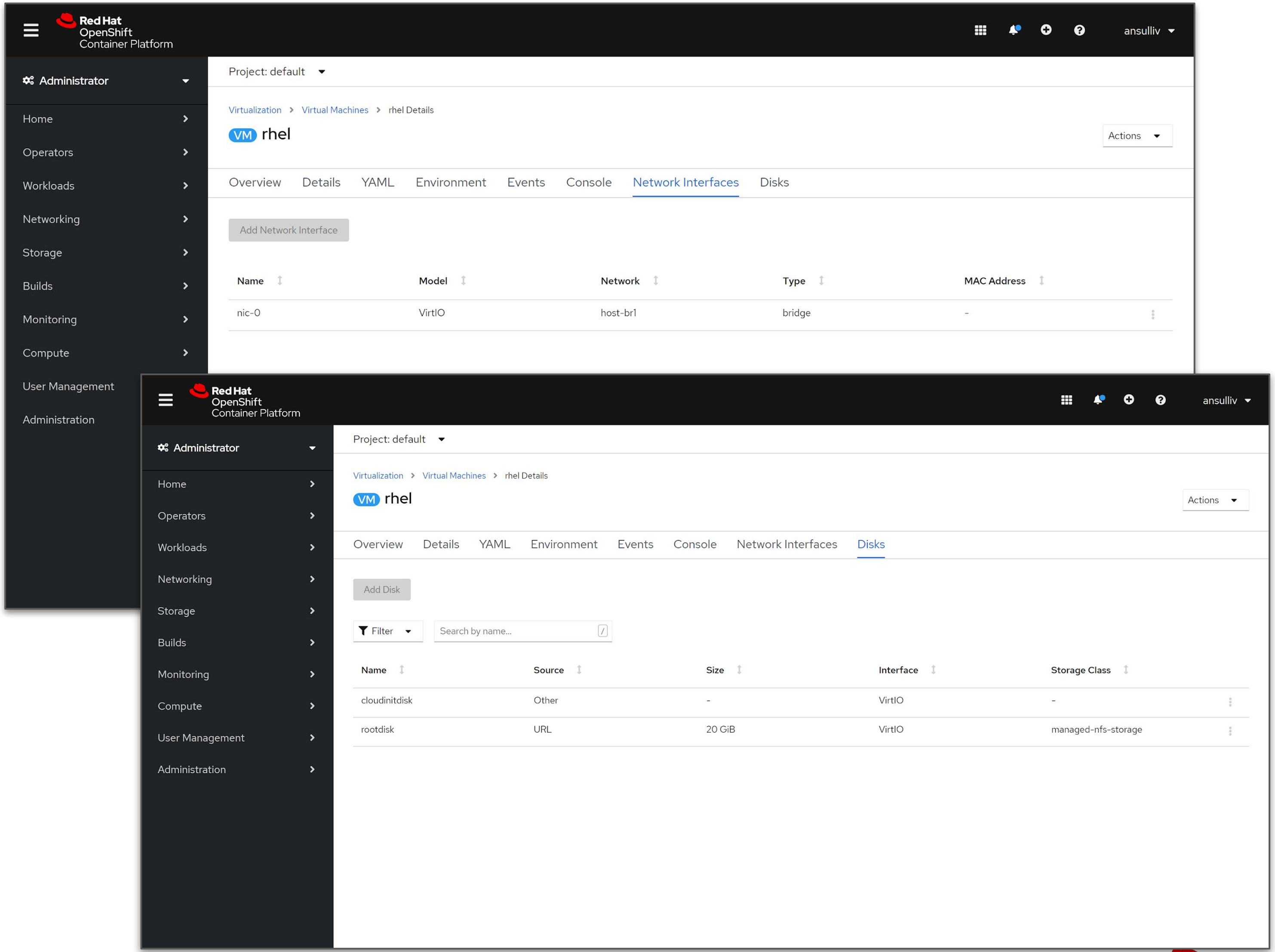
Virtual Machine - Console

- Browser-based access to the serial and graphical console of the virtual machine
- Access the console using native OS tools, e.g. `virt-viewer`, using the `virtctl` CLI command
  - `virtctl console vmname`
  - `virtctl vnc vmname`



Virtual Machine - Disks and NICs

- Add, edit, and remove NICs and disks for non-running virtual machines

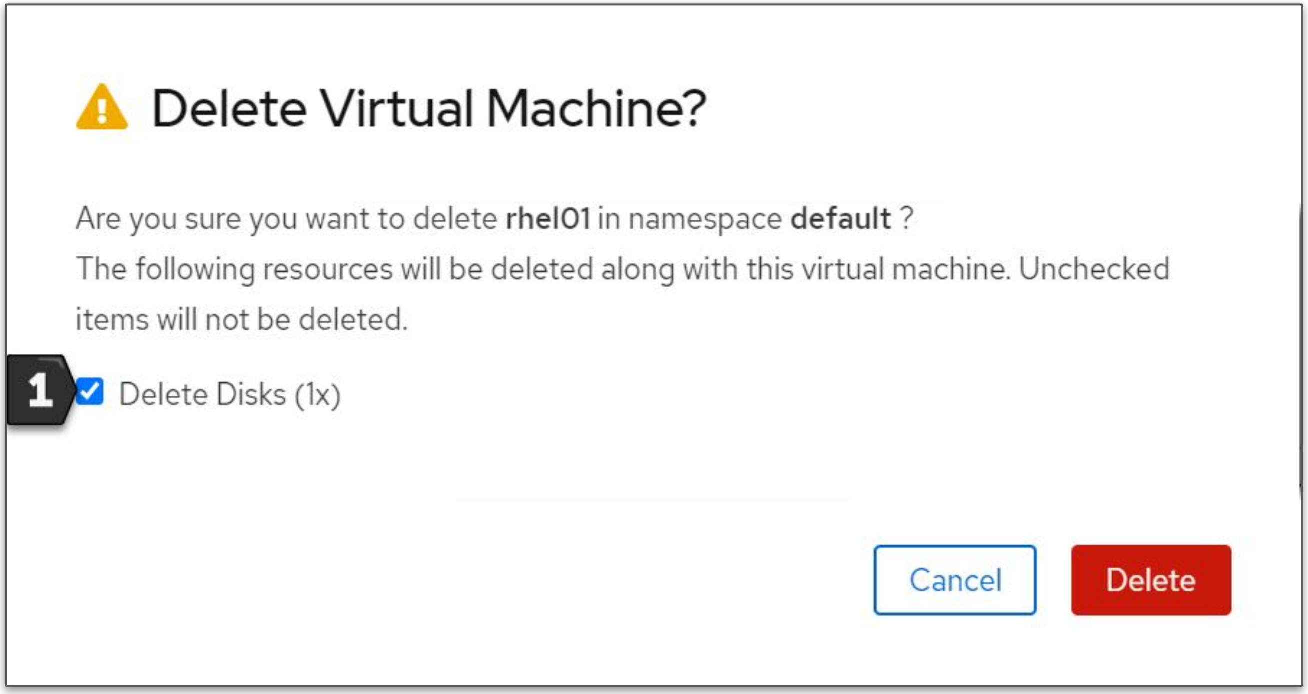


Destroy VMs

Red Hat

Destroying a Virtual Machine

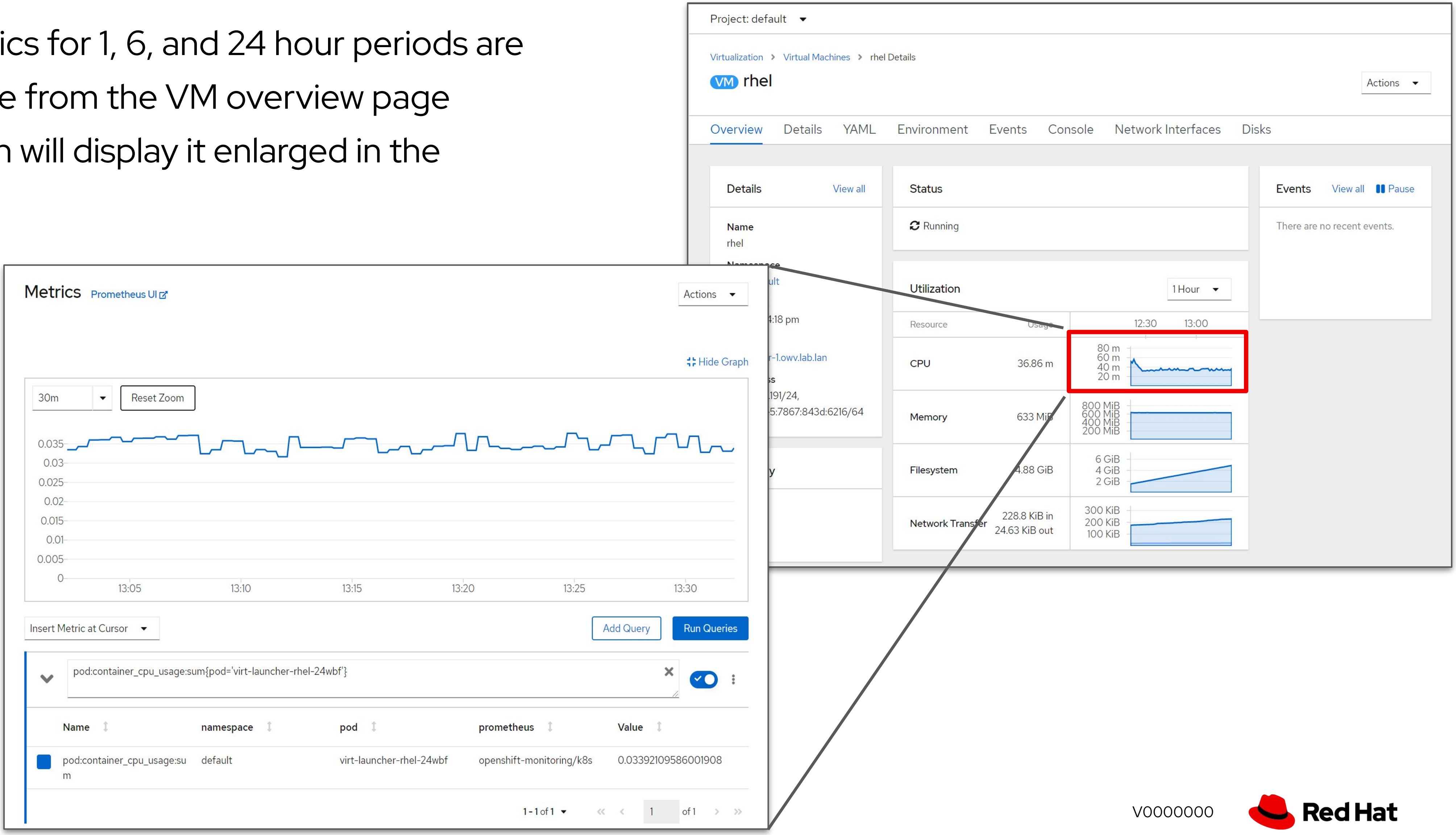
- Deleting a VM removes the VM definition
  - Optionally delete PVC-backed disks associated with the VM
- Running VMs are terminated first
- Other associated resources, e.g. Services, are not affected



Metrics

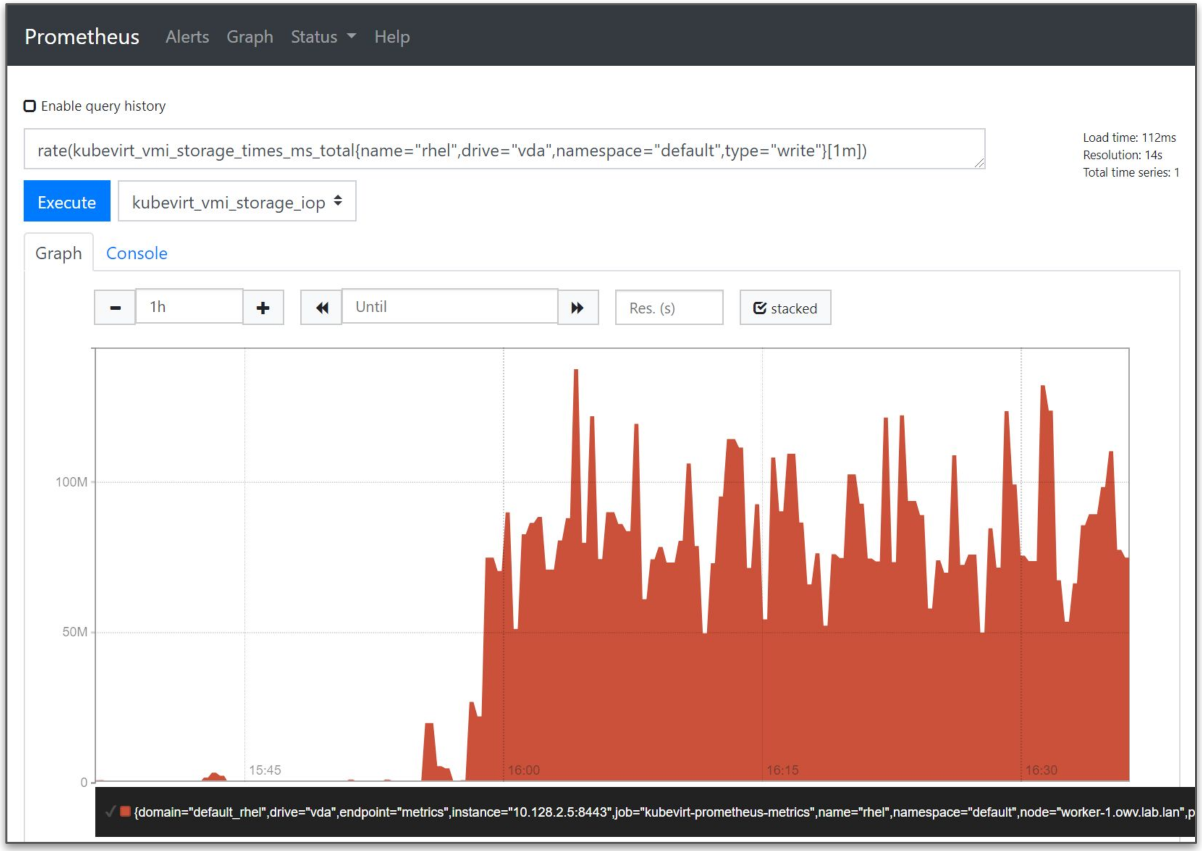
Overview Virtual Machine metrics

- Summary metrics for 1, 6, and 24 hour periods are quickly viewable from the VM overview page
- Clicking a graph will display it enlarged in the metrics UI



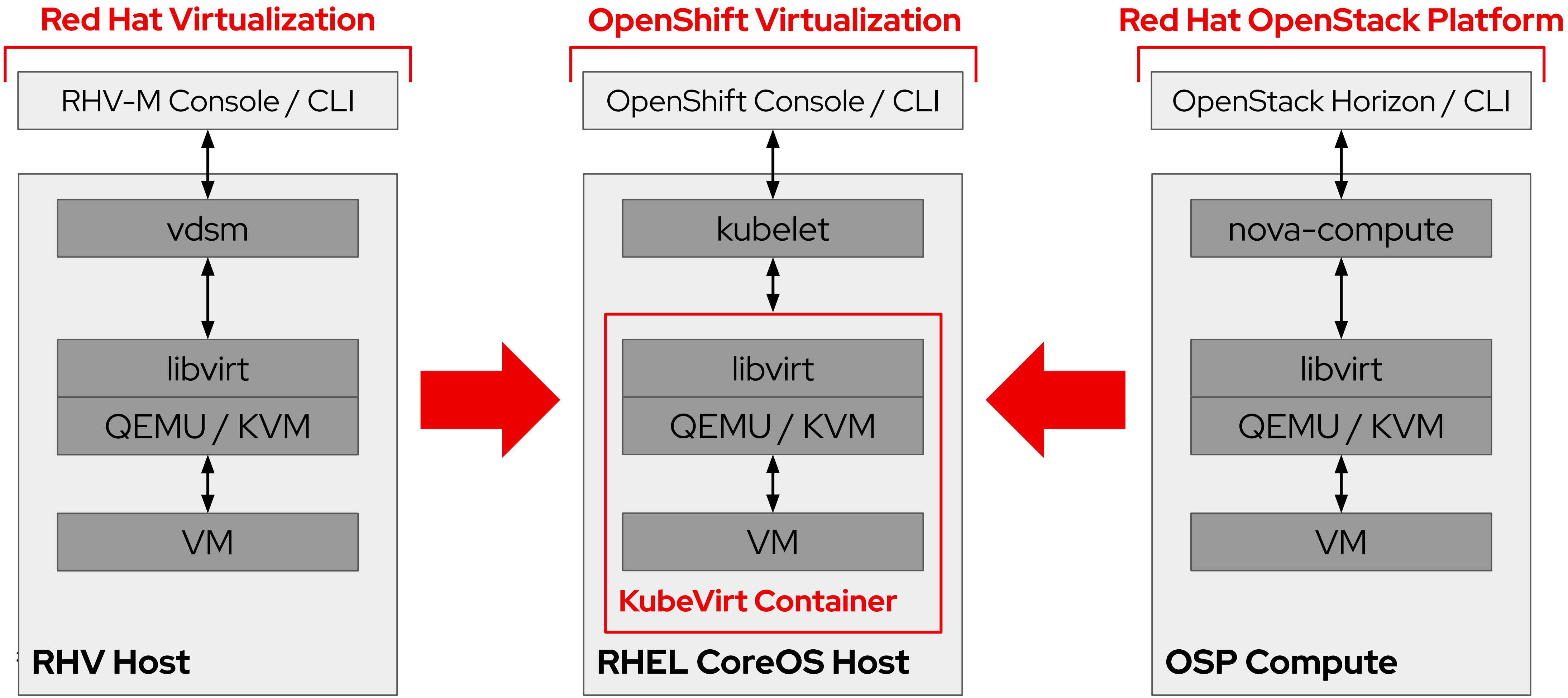
Detailed Virtual Machine metrics

- Virtual machine, and VM pod, metrics are collected by the OpenShift metrics service
  - Available under the `kubevirt` namespace in **Prometheus**
- Available per-VM metrics include
  - Active memory
  - Active CPU time
  - Network in/out errors, packets, and bytes
  - Storage R/W IOPS, latency, and throughput
- VM metrics are for VMs, not for VM pods
  - Management overhead not included in output
  - Look at virt-launcher pod metrics for
- No preexisting Grafana dashboards

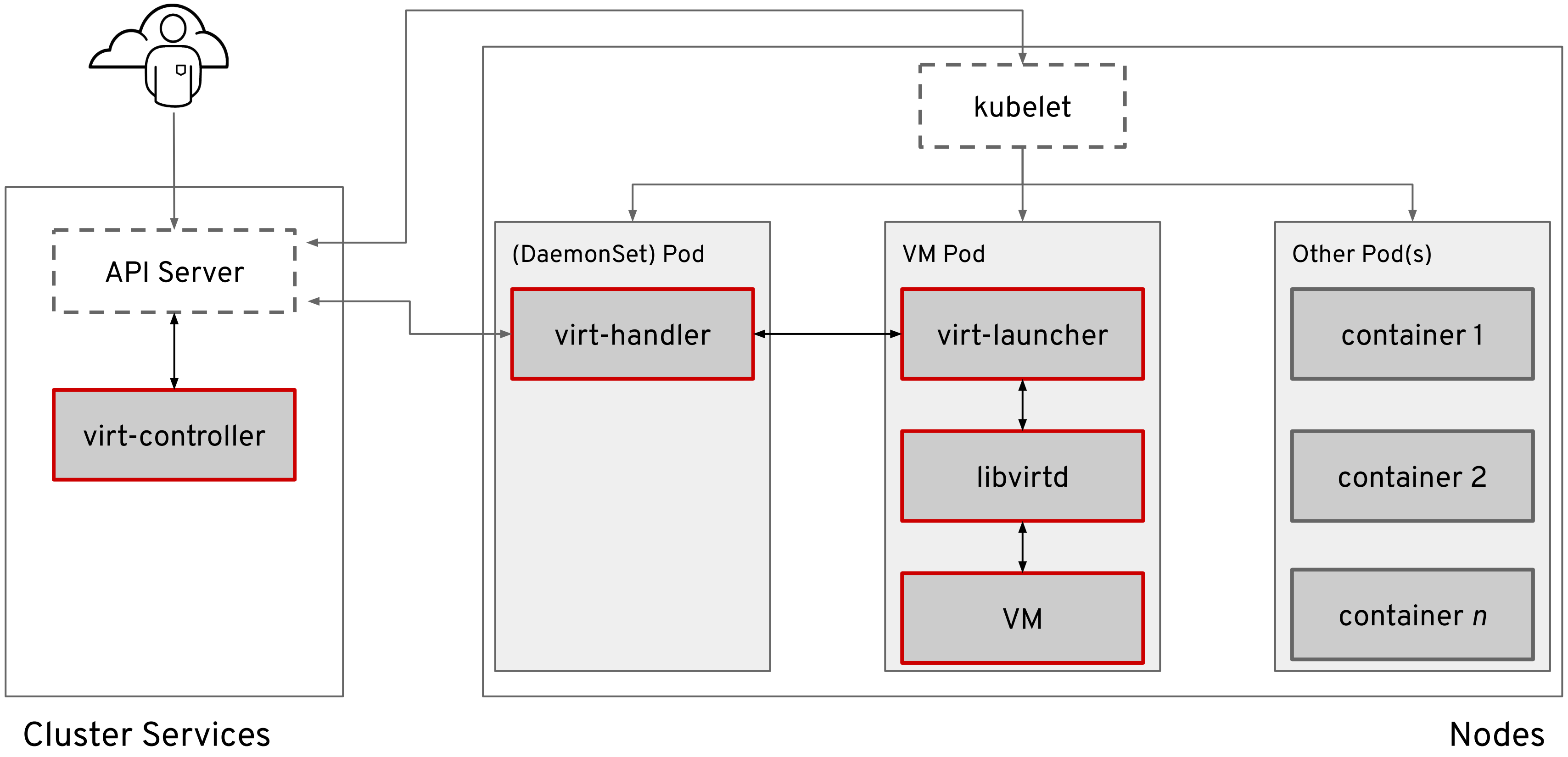


Deeper into the technology

Containerizing KVM



Architectural Overview



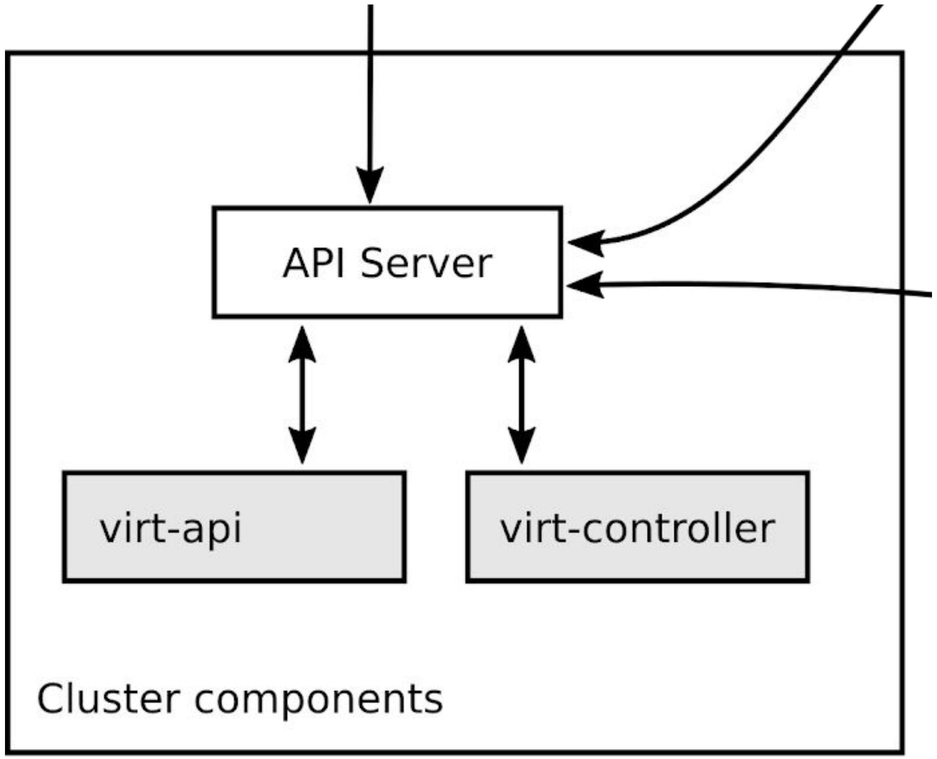
Adding virtualization to the Kubernetes API

CRD and aggregated API servers

- These are the ways to extend the Kubernetes API in order to support new entities
- For users, the new entities are indistinguishable from native resources

Single API entry point for all workloads

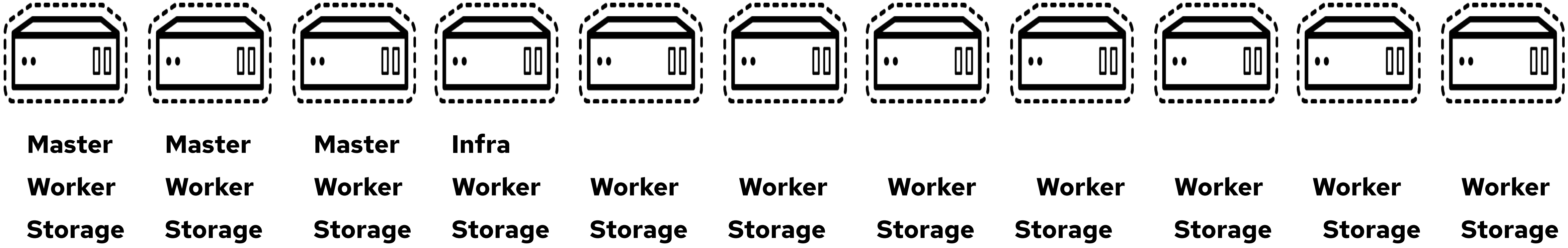
- All workloads (containers, VMs, and serverless) are managed through a single API



Openshift virtualization cluster architecture options

Red Hat

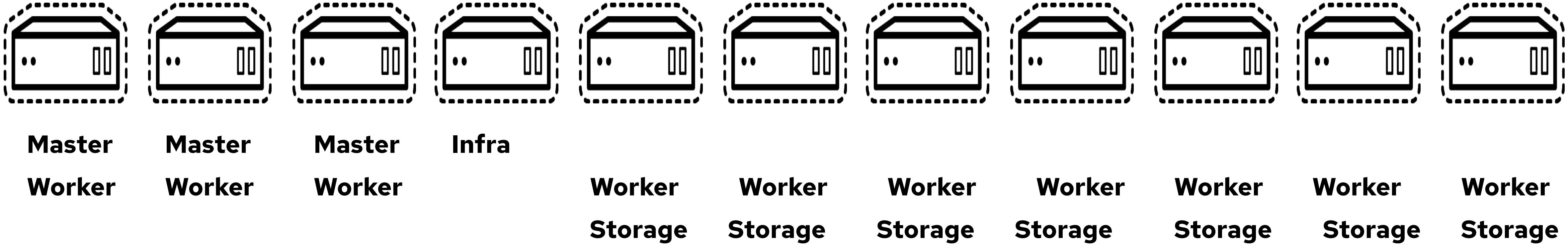
OpenShift cluster architecture #1



Everything everywhere - all 8 nodes are "workers"

- Create the cluster with the control plane as schedulable
- No dedicated infra nodes, no dedicated ODF(OCS) nodes
- Pros: no wasted resources
- Cons: must pay for all cores of all nodes, extra effort should be taken to ensure pods have appropriate QoS to prevent resource contention exacerbating performance problems

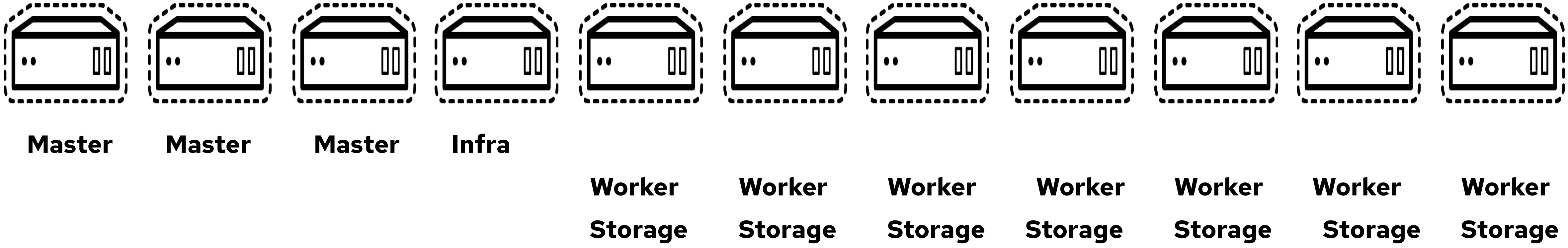
OpenShift cluster architecture #2



Shared control plane, dedicated + combined infra

- Schedulable control plane
- Dedicated infra nodes for registry, logging, metrics, and ODF(OCS)
- Pros: don't have to pay for infra node licenses
- Cons: care needs to be taken to size nodes appropriately to not strand resources, e.g. "infra nodes are only 15% utilized, but we can't put workload on those nodes without paying for the OCP entitlements"

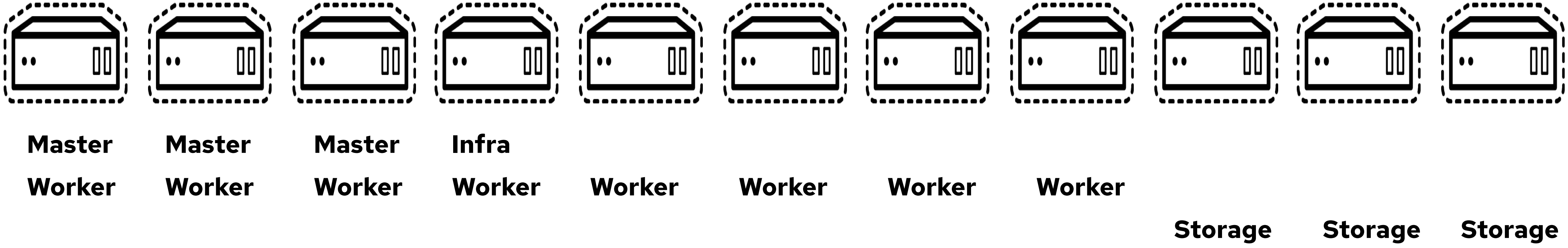
OpenShift cluster architecture #3



Dedicated control plane, dedicated infra

- Non-schedulable control plane
- Dedicated infra nodes for registry, logging, metrics, and ODF(OCS)
- Pros: control plane resource isolation prevents contention from causing performance ripples
- Cons: control plane nodes will almost certainly be dramatically under utilized, minimum 6 dedicated nodes (3 control plane, 3 infra)

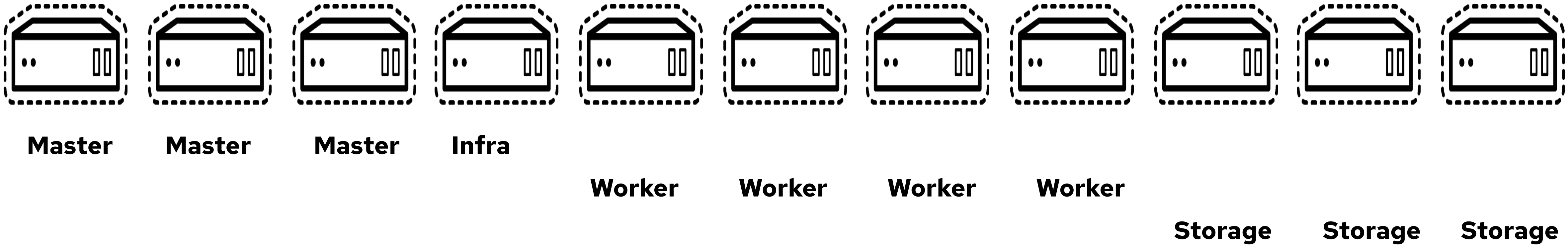
OpenShift cluster architecture #4



Shared control plane/worker/infra, dedicated ODF(OCS)

- Scheduleable control plane, no dedicated infra
- Pros: isolates OCS for performance/scale reasons
- Cons: same as above - care needs to be taken to protect control plane workloads, must pay for infra cores

OpenShift cluster architecture #5



Dedicated everything

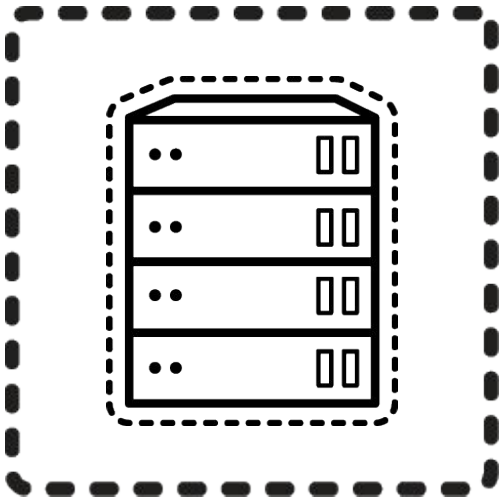
- Dedicated control plane, infra, and ODF(OCS) nodes
- Pros: lots of isolation and protection for workloads
- Cons: lots of potentially wasted resources (node right sizing is important!) and lots of nodes needed: 3 control plane, 3 ODF(OCS), 2 infra, + workers

Deep Dive on Virtual machine Resources

Red Hat

Containerized virtual machines

- Inherit many features and functions from Kubernetes
  - Scheduling, high availability, attach/detach resources
- Containerized virtual machines have the same characteristics as non-containerized
  - CPU, RAM, etc. limitations dictated by libvirt and QEMU
  - Linux and Windows guest operating systems
- Storage
  - Use Persistent Volumes Claims (PVCs) for VM disks
  - Containerized Data Importer (CDI) import VM images
- Network
  - Inherit pod network by default
  - Multus enables direct connection to external network



Red Hat OpenShift

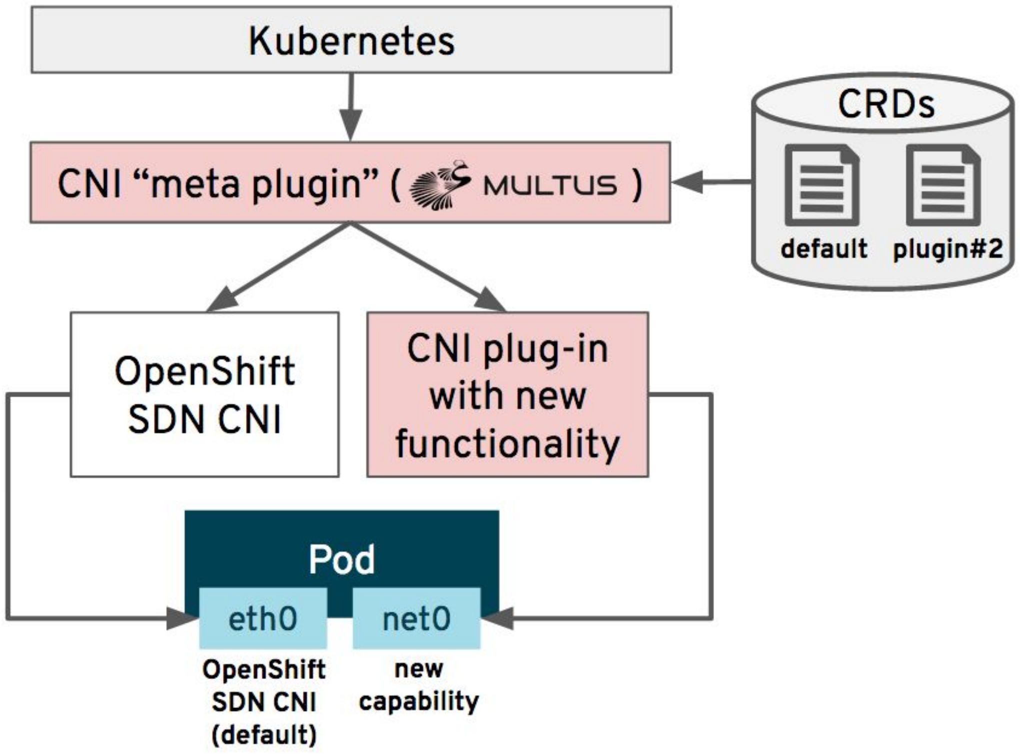
Virtual Machine Instances

- Fully based on Operators and Custom Resource Definitions (CRDs)
- A VirtualMachine (VM) CRD represents a VM definition
- A VirtualMachineInstance (VMI) CRD represents a running virtual machine
- The VM definition is optional, a VMI can be created directly
  - Can be used with standard network and storage connections
  - If persisting the VMI disks, a DataVolume is highly encouraged to prevent the VMI from launching before the import is done

Network

Virtual Machine Networking

- Virtual machines optionally connect to the standard pod network
  - OpenShift SDN, OVNKubernetes, etc.
- Additional network interfaces accessible via Multus:
  - Bridge, SR-IOV
  - VLAN and other networks can be created using nmstate at the host level
- When using at least one interface on the default SDN, Service, Route, and Ingress configuration applies to VM pods the same as others

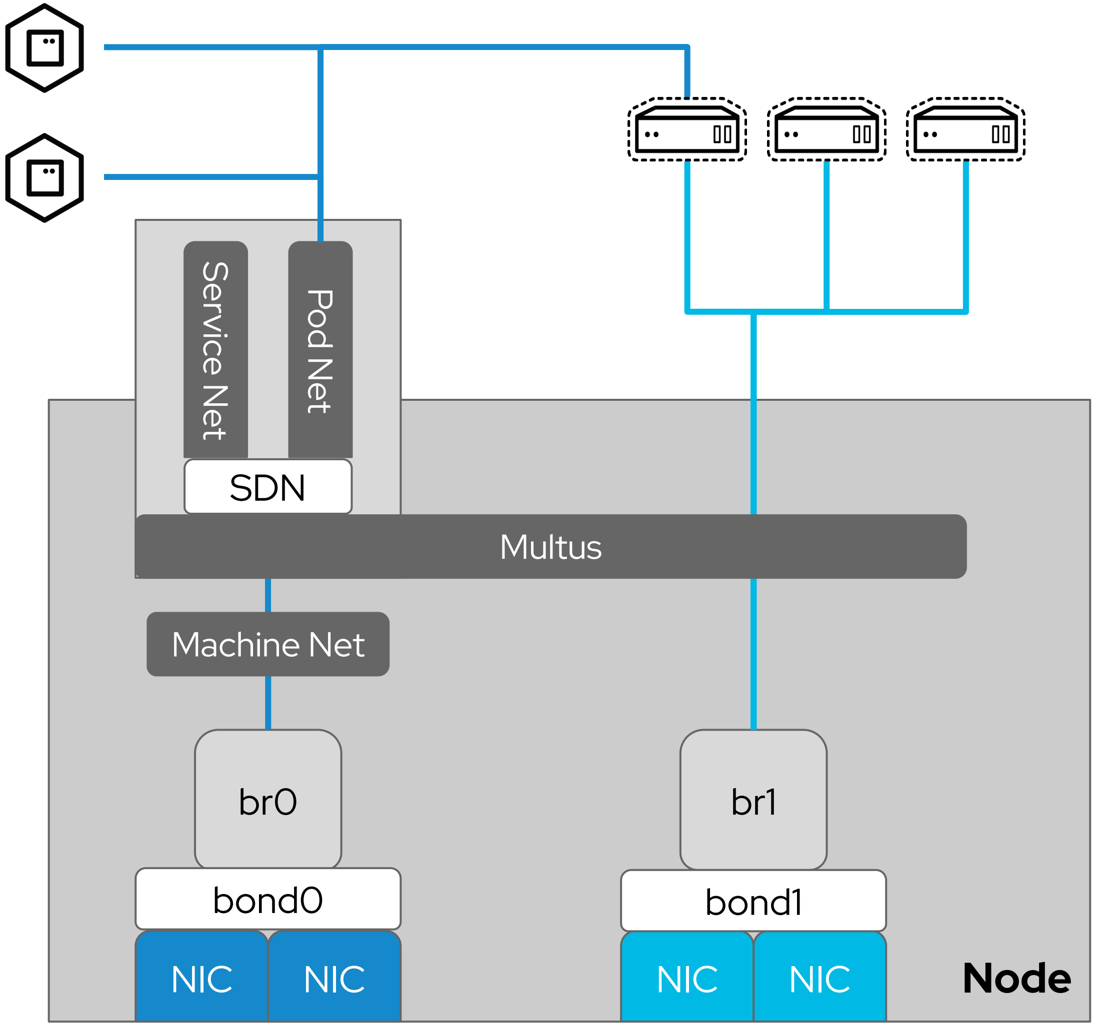


V0000000 Red Hat

Example host network configuration

- Pod, service, and machine network are configured by OpenShift automatically
  - Use kernel parameters (dracut) for configuration at install
- Use `kubernetes-nmstate`, via the nmstate Operator, to configure additional host network interfaces
  - `bond1` and `br1` in the example to the right
- VM pods connect to one or more networks simultaneously

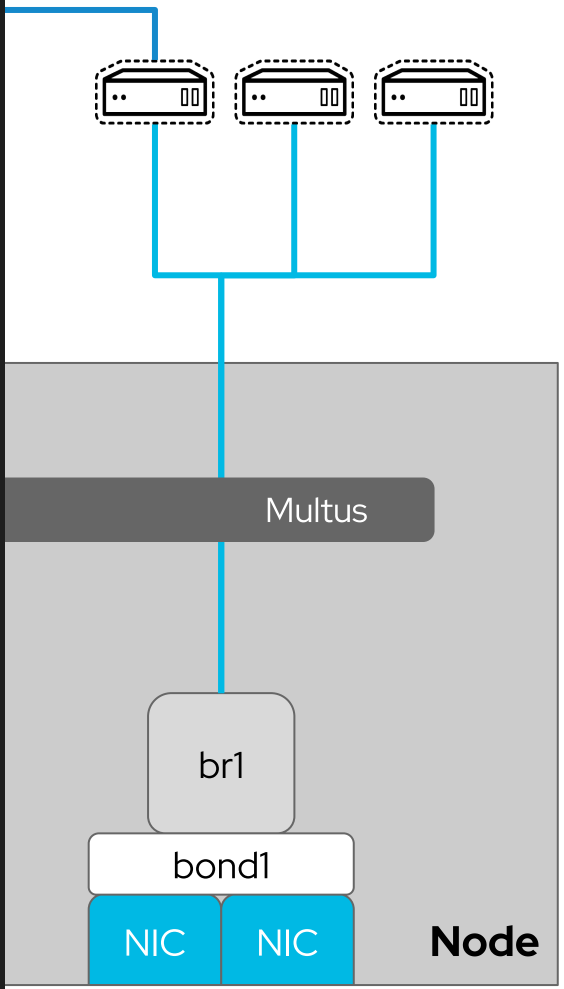
The following slides show an example of how this setup is configured



Host bond configuration

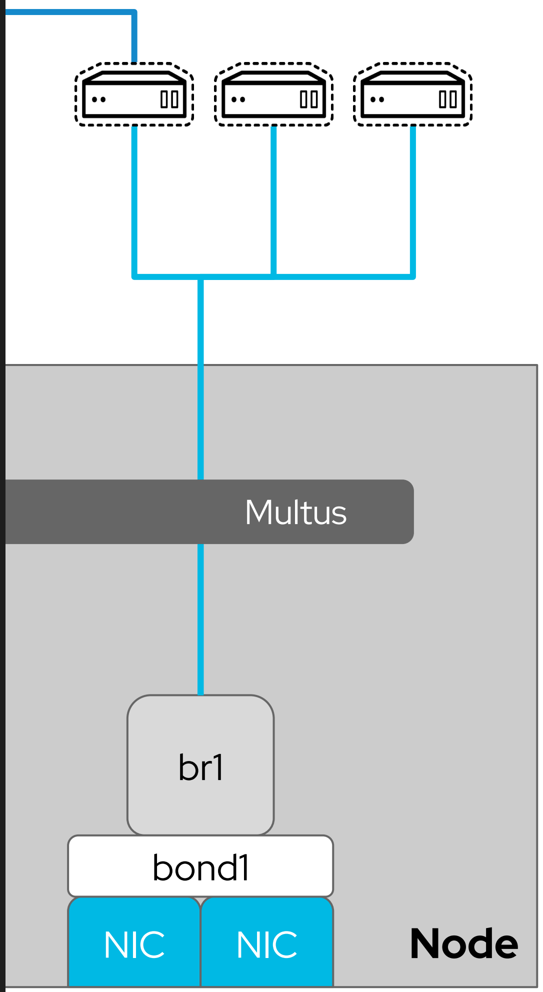
- NodeNetworkConfiguration-Policy (NNCP)
  - Nmstate operator CRD
  - Configure host network using declarative language
- Applies to all nodes specified in the `nodeSelector`, including newly added nodes automatically
- Update or add new NNCPs for additional host configs

```yaml
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: worker-bond1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  desiredState:
    interfaces:
    - name: bond1
      type: bond
      state: up
      ipv4:
        enabled: false
      link-aggregation:
        mode: balance-alb
        options:
          miimon: '100'
        slaves:
        - eth2
        - eth3
      mtu: 1450
```



Host bridge configuration

```yaml
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: worker-bond1-br1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  desiredState:
    interfaces:
      - name: br1
        description: br1 with bond1
        type: linux-bridge
        state: up
        ipv4:
          enabled: false
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: bond1
```



Host network status

- Use the `NodeNetworkConfigurationEnactment` (NNCE) object to view status of NNCP application
- Further details of the node network state can be seen using the NodeNetworkState CRD
  - oc get nns/node-name -o yaml

```
API Version:  nmstate.io/v1alpha1
Kind:         NodeNetworkConfigurationEnactment
Name:         worker-1.owv.lab.lan.worker-br1-bond1
Status:
  Conditions:
    Last Hearbeat Time:    2020-07-08T20:15:46Z
    Last Transition Time:  2020-07-08T20:15:46Z
    Message:               successfully reconciled
    Reason:                SuccessfullyConfigured
    Status:                True
    Type:                  Available
  Desired State:
    Interfaces:
      Bridge:
        Options:
          Stp:
            Enabled:  false
        Port:
          Name:     bond1
      Description:  br1 with bond1
      ipv4:
        Enabled:      false
      Name:           br1
      State:          up
      Type:           linux-bridge
```

Connecting Pods to networks

- Multus uses CNI network definitions in the NetworkAttachmentDefinition to allow access
  - `Net-attach-def` are namespaced
  - Pods cannot connect to a `net-attach-def` in a different namespace
- `cnv-bridge` and `cnv-tuning` types are used to enable VM specific functions
  - MAC address customization
  - MTU and promiscuous mode
  - sysctls, if needed

- Pod connections are defined using an annotation
  - Pods can have many connections to many networks

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: br1-public
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "br1-public",
    "plugins": [
      {
        "type": "cnv-bridge",
        "bridge": "br1"
      },
      {
        "type": "cnv-tuning"
      }
    ]
  }'
```

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: application-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: bond1-br1
```

Connecting VMs to networks

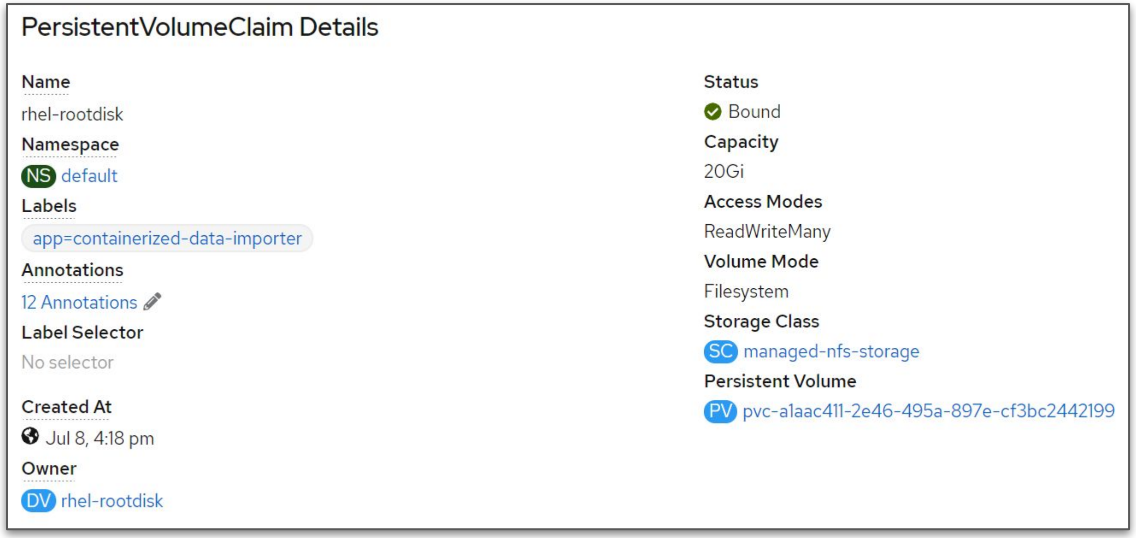
- Virtual machine interfaces describe NICs attached to the VM
  - `spec.domain.devices.interfaces`
  - Model: virtio, e1000, pcnet, rtl8139, etc.
  - Type: masquerade, bridge
  - MAC address: customize the MAC
- The networks definition describes the connection type
  - `spec.networks`
  - Pod = default SDN
  - Multus = secondary network using Multus
- Using the GUI makes this simple and removes the need to edit / manage connections in YAML

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
  name: demo-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
              model: virtio
              name: nic-0
      hostname: demo-vm
      networks:
        - multus:
            networkName: bond1-br1
          name: nic-0
```

Storage

Virtual Machine Storage

- OpenShift Virtualization uses the Kubernetes PersistentVolume (PV) paradigm
- PVs can be backed by
  - In-tree iSCSI, NFS
  - CSI drivers
  - Local storage using host path provisioner
  - ODF/OpenShift Container Storage
- Dynamically or statically provisioned PVs
- RWX required for live migration
- Disks are attached using VirtIO or SCSI controllers
  - Connection order defined in the VM definition
- Boot order customized via VM definition



VM disks in PVCs

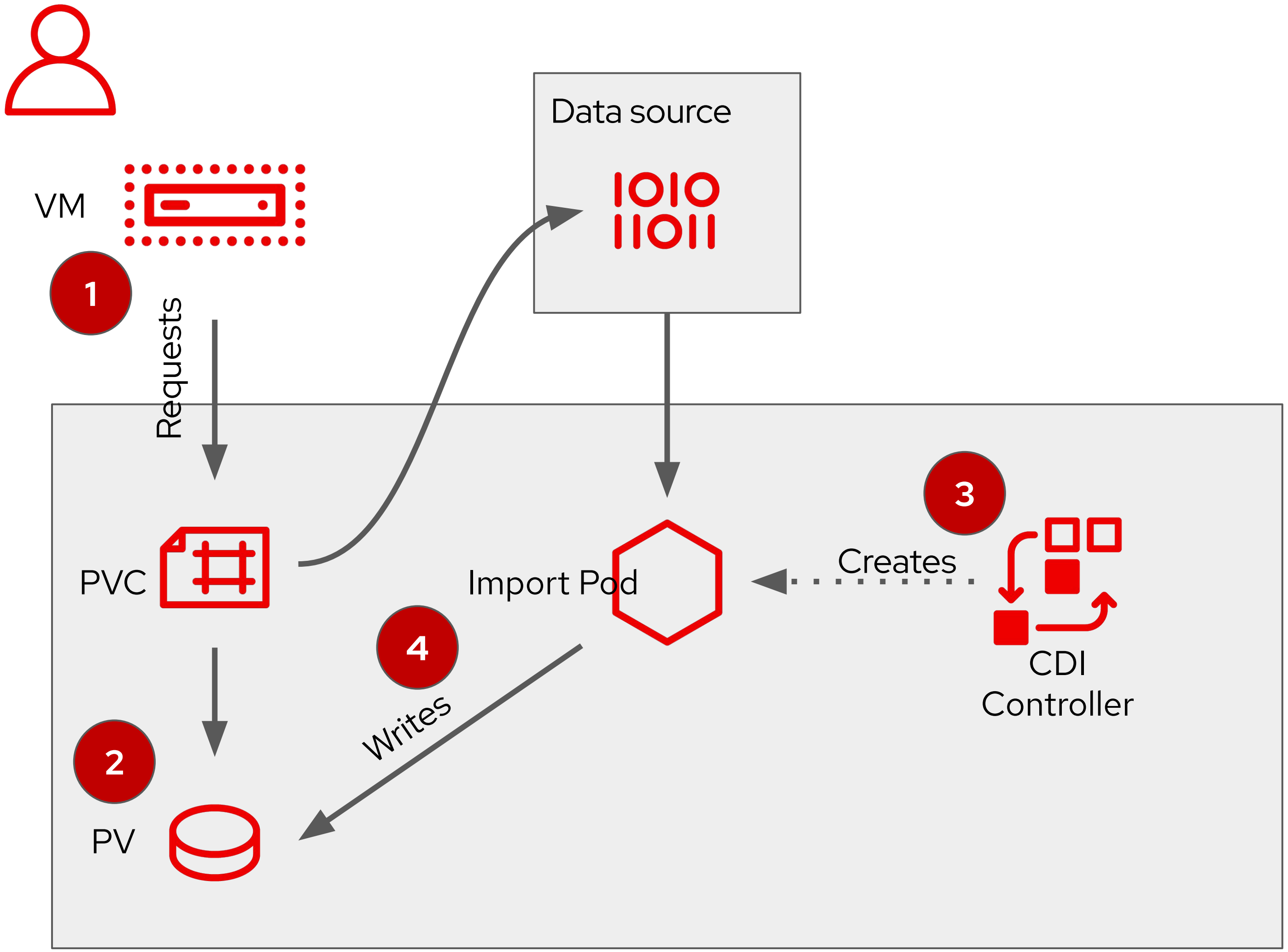
- VM disks on FileSystem PVCs are created as thin provisioned raw images
  - Thick provisioned disks are not created by CDI, may be possible manually
- Block PVCs are attached directly to the VM
- CSI operations, e.g. snapshot and clone, are not supported with VM disk PVCs
  - Use DataVolumes to clone VM disks
- PVC resize does not modify the size of the VM disk
  - Not currently supported
- Hot add is not supported (for any virtual hardware)

DataVolumes

- VM disks can be imported from multiple sources using DataVolumes, e.g. an HTTP(S) or S3 URL for a QCOW2 or raw disk image, optionally compressed
- DataVolumes are created view explicit object definition or as a part of the VM definition
- DataVolumes use the `ContainerizedDataImporter` to connect, download, and prepare the image for OpenShift Virtualization
- DataVolumes create PVCs based on defaults defined in the `kubevirt-storage-class-defaults` ConfigMap

```yaml
dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1alpha1
    kind: DataVolume
    metadata:
      creationTimestamp: null
      name: vm-rootdisk
    spec:
      pvc:
        accessModes:
          - ReadWriteMany
        resources:
          requests:
            storage: 20Gi
        storageClassName: my-storage-class
        volumeMode: Filesystem
      source:
        http:
          url: 'http://web.server/disk-image.qcow2'
```

Containerized Data Importer



1. The user creates a virtual machine with a DataVolume
2. The StorageClass is used to satisfy the PVC request
3. The CDI controller creates an importer pod, which mounts the PVC and retrieves the disk image. The image could be sourced from S3, HTTP, or other accessible locations
4. After completing the import, the import pod is destroyed and the PVC is available for the VM

Ephemeral Virtual Machine Disks

- VMs booted via PXE or using a container image can be "diskless"
  - PVCs may be attached and mounted as secondary devices for application data persistence
- VMs based on container images use the standard copy-on-write graph storage for OS disk R/W
  - Consider and account for capacity and IOPS during RHCOS disk sizing if using this type
- An `emptyDisk` may be used to add additional ephemeral capacity for the VM

```yaml
spec:
  domain:
    disks:
      - bootOrder: 1
        disk:
          bus: virtio
        name: rootdisk
  volumes:
    - containerDisk:
        image: registry.lab.lan:5000/fedora:31
      name: rootdisk
```

Helper disks

- OpenShift Virtualization attaches disks to VMs for injecting data
  - Cloud-Init
  - ConfigMap
  - Secrets
  - ServiceAccount
- These disks are read-only and can be mounted by the OS to access the data within

```yaml
spec:
  domain:
    devices:
      - disk:
          bus: virtio
        name: cloudinitdisk
  volumes:
    - cloudInitNoCloud:
        userData: |-
          #cloud-config
          password: redhat
          chpasswd: { expire: False }
      name: cloudinitdisk
```

Name	Source	Size	Interface	Storage Class
cloudinitdisk	Other	-	VirtIO	-

Comparing with traditional virtualization platforms

Live Migration

- Live migration moves a virtual machine from one node to another in the OpenShift cluster
- Can be triggered via GUI, CLI, API, or automatically
- RWX storage is required, cannot use bridge connection to pod network
- Live migration is cancellable by deleting the API object
- Default maximum of five (5) simultaneous live migrations
  - Maximum of two (2) outbound migrations per node, 64MiB/s throughput each

Migration Reason	vSphere	RHV	OpenShift Virtualization
Resource contention	DRS	Cluster policy	Pod eviction policy, pod descheduler
Node maintenance	Maintenance mode	Maintenance mode	Maintenance mode, node drain

Automated live migration

- OpenShift / Kubernetes triggers pod rebalance actions based on multiple factors
    - Pod rebalance applies to VM pods equally and will result in a live migration
- Eviction policies
    - Soft
    - Hard
- Pod descheduler
- Pod disruption policy

VM scheduling

- VM scheduling follows pod scheduling rules
  - Node selectors
  - Taints / tolerations
  - Pod and node affinity / anti-affinity
- Kubernetes scheduler takes into account many additional factors
  - Resource load balancing - requests and reservations
  - CPU pinning, NUMA
  - Large / Huge page support for VM memory
- Resources are managed by Kubernetes
  - CPU and RAM requests less than limit - `Burstable` QoS by default
  - K8s QoS policy determines scheduling priority: `BestEffort` class is evicted before `Burstable` class, which is evicted before `Guaranteed` class

Red Hat

Node Resource Management

- VM density is determined by multiple factors controlled at the cluster, OpenShift Virtualization, pod, and VM levels
- Pod QoS policy
  - Burstable (limit > request) allows more overcommit, but may lead to more frequent migrations
  - Guaranteed (limit = request) enables less overcommitment, but may have less physical resource utilization on the hosts
- Cluster Resource Override Operator provides global overcommit policy, can be customized per project for additional control
- VM pods request a small amount of additional memory, used for libvirt/QEMU overhead
  - Administrator can set this to be overcommitted
- Enable kernel same-page merging (KSM) by starting the daemon using a MachineConfig

High availability

- Node failure is detected by Kubernetes and results in the pods from the lost node being rescheduled to the surviving nodes
- VMs are not scheduled to nodes which have not had a heartbeat from `virt-handler`, regardless of Kubernetes node state
- Additional monitoring may trigger automated action to force stop the VM pods, resulting in rescheduling
  - May take up to 5 minutes for `virt-handler` and/or Kubernetes to detect failure
  - Liveness and Readiness probes may be configured for VM-hosted applications

Terminology comparison

Feature	RHV	OpenShift Virtualization	vSphere
Where VM disks are stored	Storage Domain	PVC	datastore
Policy based storage selection	None	StorageClass	SPBM
Non-disruptive VM migration	Live migration	Live migration	vMotion
Non-disruptive VM storage migration	Storage live migration	N/A	Storage vMotion
Active resource balancing	Cluster scheduling policy	Pod eviction policy, descheduler	Dynamic Resource Scheduling (DRS)
Physical network configuration	Host network config (via nmstate w/4.4)	nmstate Operator, Multus	vSwitch / DvSwitch
Overlay network configuration	OVN	OCP SDN (OpenShiftSDN, OVNKubernetes, and partners), Multus	NSX-T
Host / VM metrics	Data warehouse + Grafana (RHV 4.4)	OpenShift Metrics, health checks	vCenter, vROps

Runtime awareness

Deploy and configure

- OpenShift Virtualization is deployed as an Operator utilizing multiple CRDs, ConfigMaps, etc. for primary configuration
- Many aspects are controlled by native Kubernetes functionality
    - Scheduling
    - Overcommitment
    - High availability
- Utilize standard Kubernetes / OpenShift practices for applying and managing configuration

Compute configuration

- VM nodes should be physical with CPU virtualization technology enabled in the BIOS
  - Nested virtualization *works*, but is not supported
  - Emulation *works*, but is not supported (and is extremely slow)
- Node labeler detects CPU type and labels nodes for compatibility and scheduling
- Configure overcommitment using native OpenShift functionality - Cluster Resource Override Operator
  - Optionally, customize the default project so that non-VM pods are not overcommitted
  - Customize projects hosting VMs for overcommit policy
- Enable KSM using MachineConfig, ballooning is not supported
- Apply Quota and LimitRange controls to projects with VMs to manage resource consumption

Network configuration

- Apply traditional network architecture decision framework to OpenShift Virtualization
  - Resiliency, isolation, throughput, etc. determined by combination of application, management, storage, migration, and console traffic
  - Most clusters are not VM only, include non-VM traffic when planning
- Node interface on the `MachineNetwork` is used for “primary” communication, including SDN
  - This interface should be both resilient and high throughput
  - Used for migration and console traffic
  - Configure this interface at install time using kernel parameters, reinstall node if configuration changes
- Additional interfaces, whether single or bonded, may be used for traffic isolation, e.g. storage and VM traffic
  - Configure using nmstate Operator, apply configuration to nodes using selectors on NNCP

Storage configuration

- Local storage may be utilized via the Host Path Provisioner
  - Local-only, non-shared storage means no live migration
- Create shared storage from local resources using ODF/OpenShift Container Storage
  - RWX file and block devices for live migration
- No preference for storage protocol, use what works best for the application(s)
- Storage backing PVs should provide adequate performance for VM workload
  - Monitor latency from within VM, monitor throughput from OpenShift
- For IP storage (NFS, iSCSI), consider using dedicated network interfaces
  - Will be used for all PVs, not just VM PVs
- Certified CSI drivers are recommended
  - No CSI snapshot integration
  - Non-certified work, but do not have same level of OpenShift testing

Deploying a VM operating system

Creating virtual machines can be accomplished in multiple ways, each offering different options and capabilities

- Start by answering the question “Do I want to manage my VM like a container or a traditional VM?”
- Deploying the OS persistently, i.e. “I want to manage like a traditional VM”
  - Methods:
    - Import a disk with the OS already installed (e.g. cloud image) from a URL or S3 endpoint using a DataVolume, or via CLI using virtctl
    - Clone from an existing PVC or VM template
  - VM state will remain through reboots and, when using RWX PVCs, can be live migrated
- Deploying the OS non-persistently, i.e. “I want to manage like a container”
  - Methods:
    - Diskless, via PXE
    - Container image, from a registry
  - VM has no state, power off will result in disk reset. No live migration.
- Import disks deployed from a container image using CDI to make them persistent

Deploying an application

Once the operating system is installed, the application can be deployed and configured several ways

- The application is pre-installed with the OS
  - This is helpful when deploying from container image or PXE as all components can be managed and treated like other container images
- The application is installed to a container image
  - Allows the application to be mounted to the VM using a secondary disk. Decouples OS and app lifecycle. When used with a VM that has a persistently deployed OS this breaks live migration
- The application is installed after OS is installed to a persistent disk
  - cloud-init - perform configuration operations on first boot, including OS customization and app deployment
  - SSH/Console - connect and administer the OS just like any other VM
  - Ansible or other automation - An extension of the SSH/console method, just automated

Additional resources

More information

- Openshift Test Drive:
  - https://www.redhat.com/en/technologies/cloud-computing/openshift/try-it
- Documentation:
  - OpenShift Virtualization: https://docs.openshift.com
  - KubeVirt: https://kubevirt.io
- Demos and video resources: http://demo.openshift.com
- Labs and workshops: coming soon to RHPDS

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat