



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
 - Network
- Comparison of Openshift virtualization with Traditional Virtualization

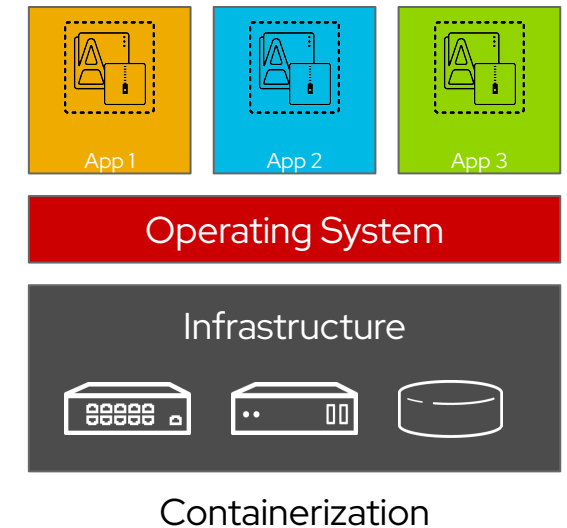
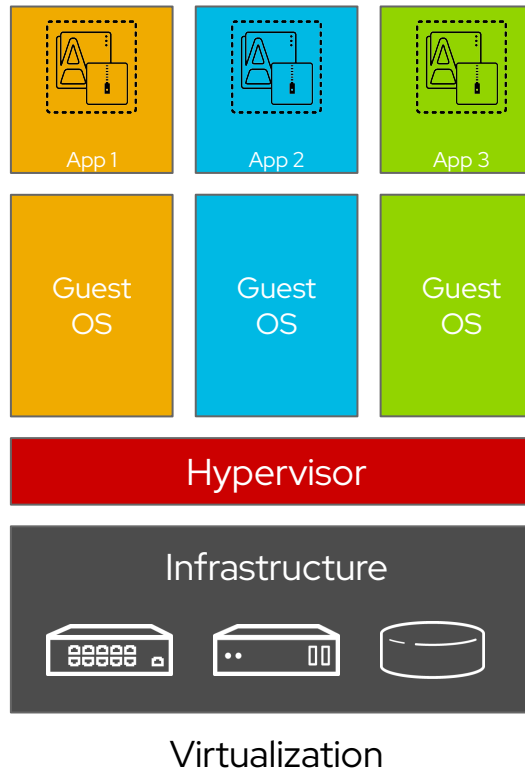
But first: Some Introductions

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

What is OpenShift Virtualization?

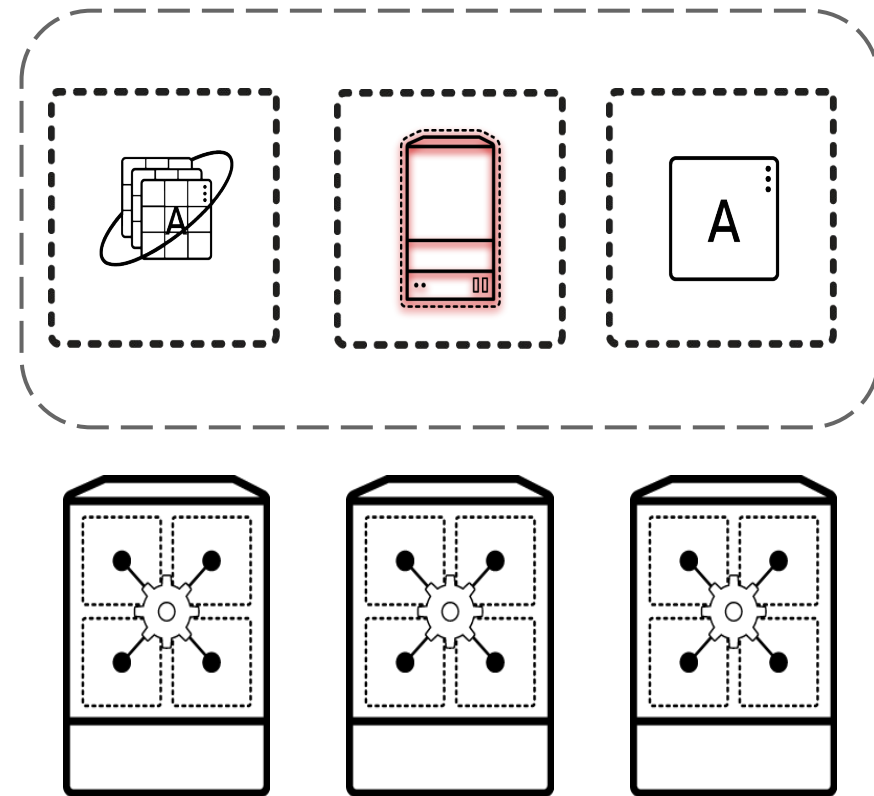
Containers are not virtual machines

- 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



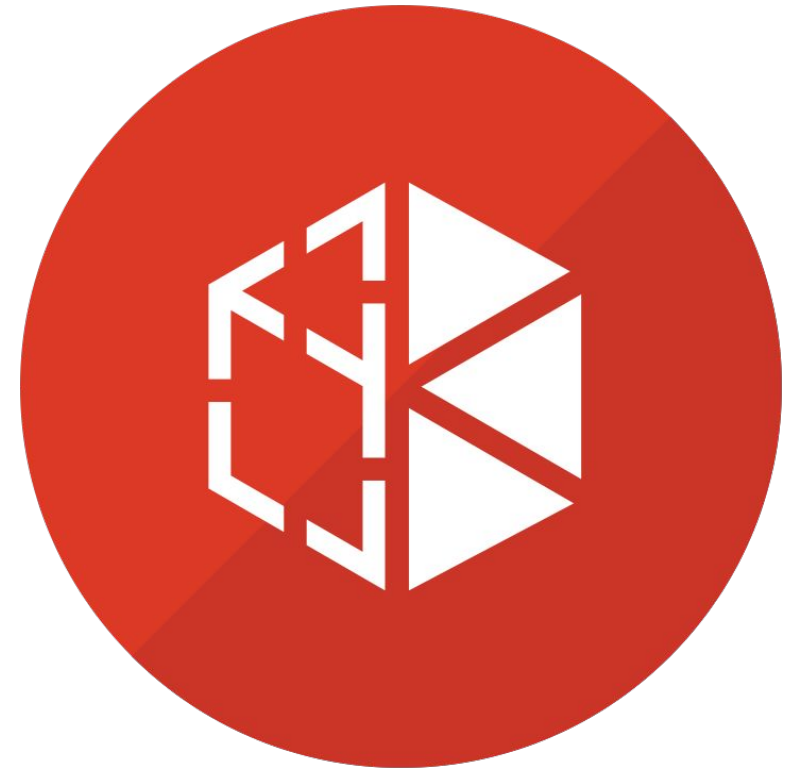
Virtual machines can be put into containers

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



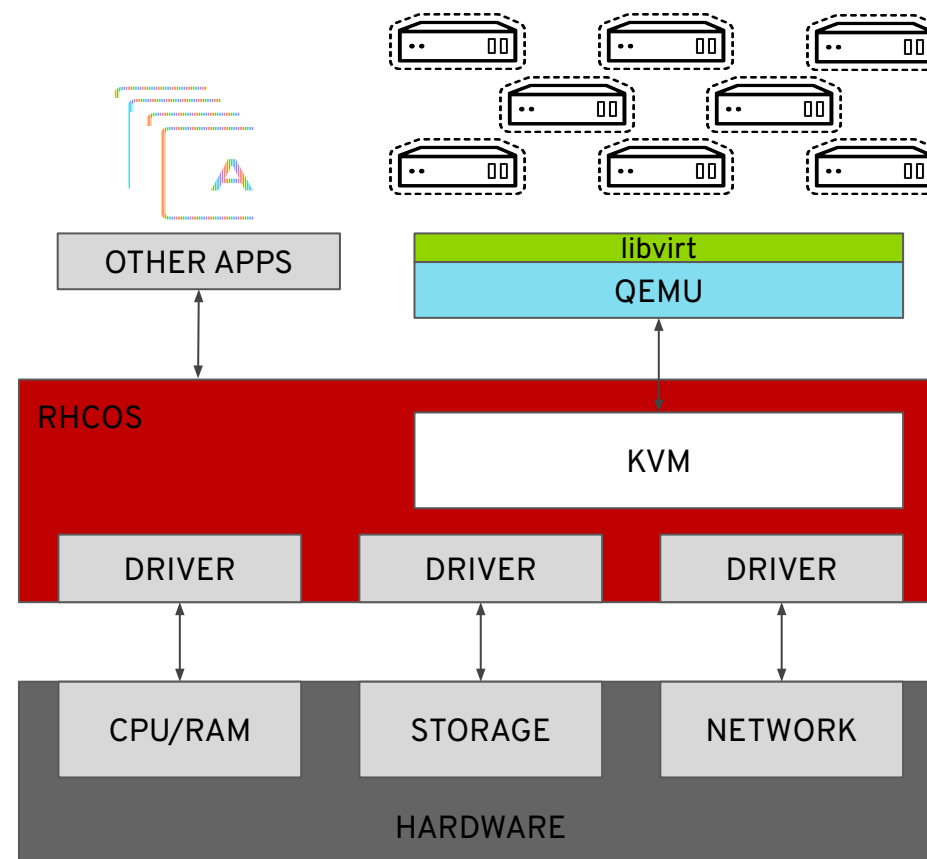
OpenShift Virtualization

- 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

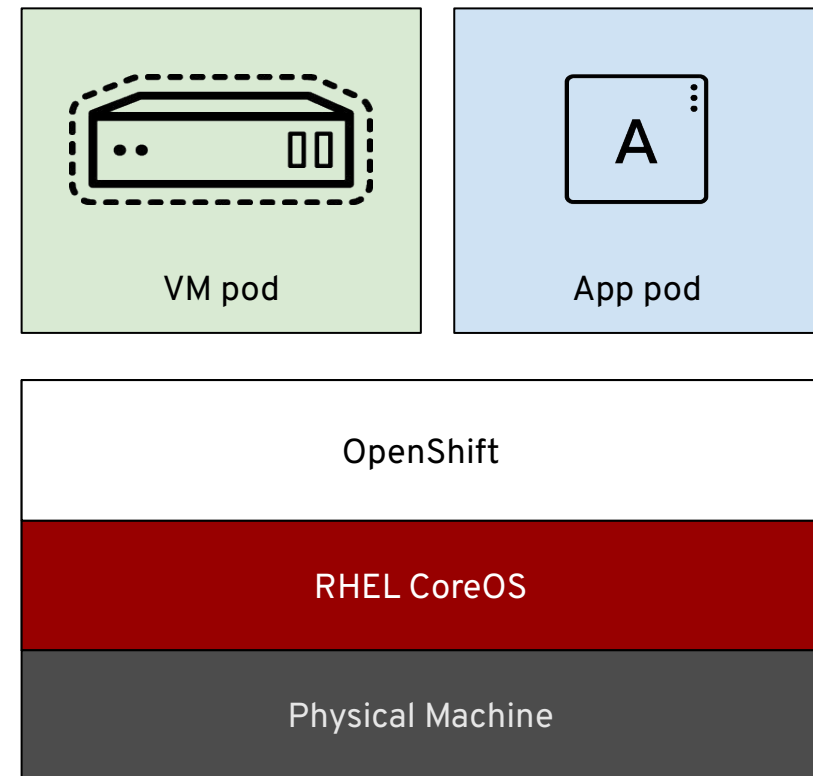
- OpenShift Virtualization uses KVM, the Linux kernel hypervisor
- KVM is a core component of the Red Hat Enterprise Linux kernel
 - KVM has 10+ years of production use: Red Hat Virtualization, Red Hat OpenStack Platform, and RHEL all leverage KVM, QEMU, and libvirt
- QEMU uses KVM to execute virtual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

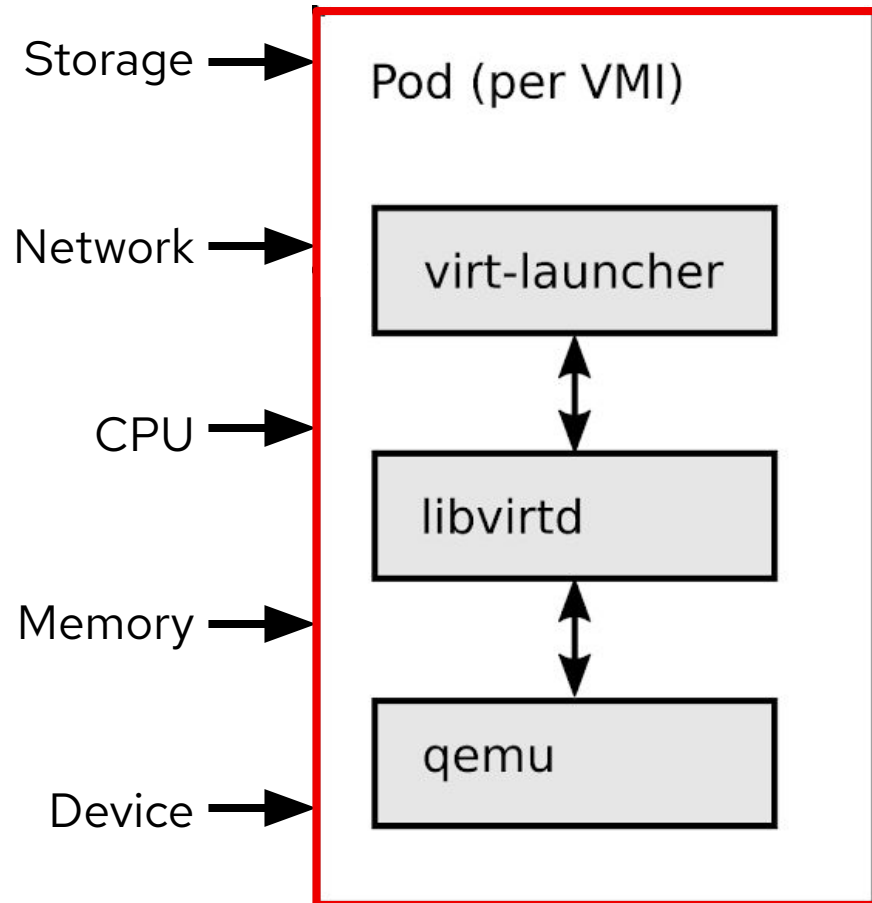


Virtualization native to Kubernetes

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

```
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

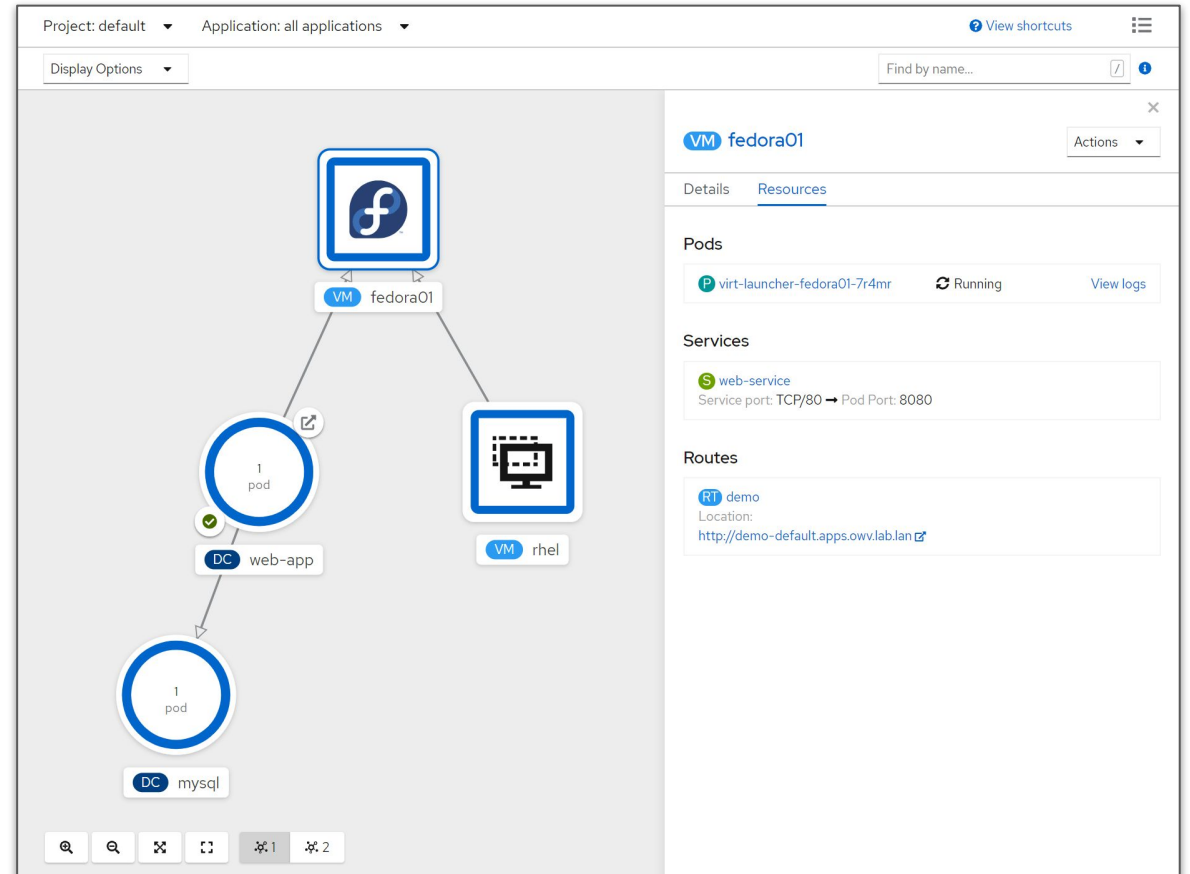
- libvirt and qemu from RHEL are mature, have high performance, provide stable abstractions, and have a minimal overhead.

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

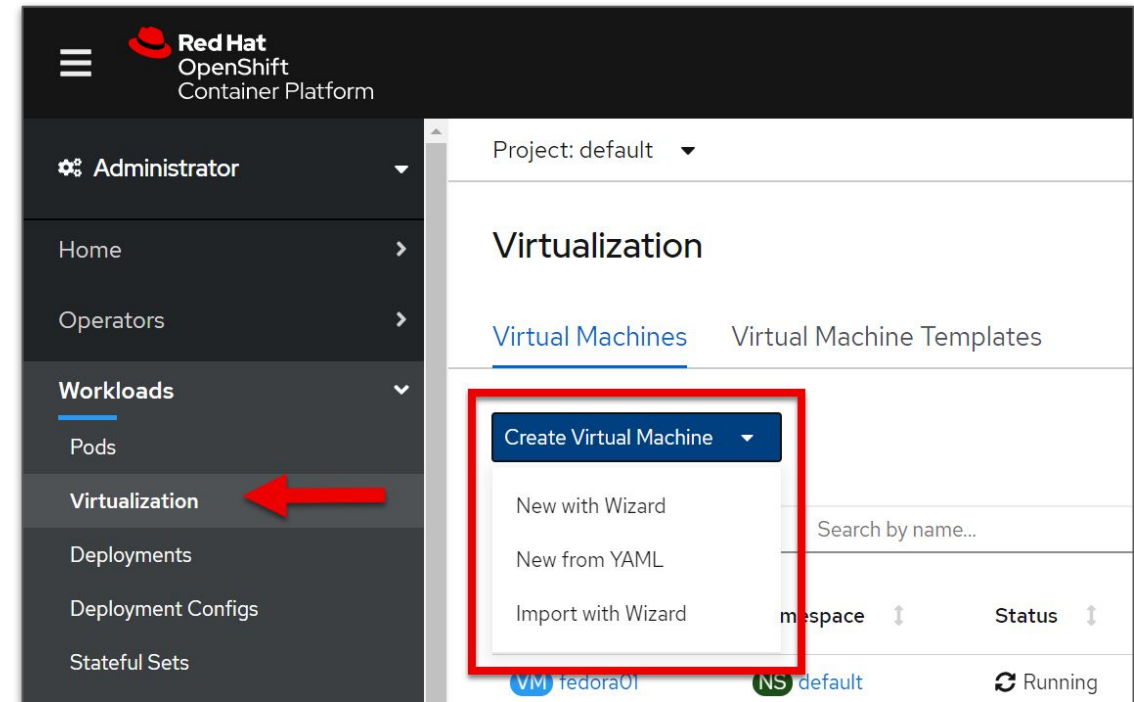
The screenshot shows the Red Hat OpenShift Container Platform web interface. The left sidebar contains navigation options: Administrator, Home, Operators, Workloads (selected), Pods, Virtualization (selected), Deployments, Deployment Configs, Stateful Sets, Secrets, Config Maps, Cron Jobs, Jobs, Daemon Sets, Replica Sets, and Replication Controllers. The main content area is titled 'Virtualization' and shows a 'Virtual Machines' tab. A 'Create Virtual Machine' button is visible. Below it is a search bar and a table of VMs.

Name	Namespace	Status	Created	Node	IP Address
VM fedora01	NS default	Running	Jul 9, 5:00 pm	worker-0.owv.lab.lan	10.131.0.74
VM rhel	NS default	Running	Jul 8, 4:18 pm	worker-0.owv.lab.lan	192.168.14.163/24, fe80::87cc:48e:1e2:9d23/64
VM rhel01	NS default	Off	Jul 9, 4:58 pm		
VM windows2019	NS default	Running	Jul 9, 5:01 pm	worker-1.owv.lab.lan	10.128.2.52

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



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

Project: default ▾

Create Virtual Machine

1 General
2 Networking
3 Storage
4 Advanced
Cloud-init
Virtual Hardware
5 Review
6 Result

Name *

Description

Template
No template available ▾

Source *

1 --- Select Source ---
--- Select Source ---
PXE
URL
Container
Disk

Flavor * ⓘ

2 Custom
--- Select Flavor ---
Tiny
Small
Medium
Large
Custom

Workload Profile * ⓘ

3 --- Select Workload Profile ---
--- Select Workload Profile ---
desktop
highperformance
server

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

The screenshot displays the 'Create Virtual Machine' interface in OpenShift. The 'Networking' step is active, showing a table of network interfaces. A modal window titled 'Add Network Interface' is open, with four numbered callouts (1-4) highlighting the configuration fields: 1 points to the 'Add Network Interface' button, 2 to the 'Model' dropdown (set to 'VirtIO'), 3 to the 'Network' dropdown (set to 'host-brl'), and 4 to the 'Type' dropdown (set to 'bridge').

Project: default ▾

Create Virtual Machine

1 General
2 **Networking**
3 Storage
4 Advanced
Cloud-init
Virtual Hardware
5 Review
6 Result

Network Interfaces Add Network Interface

Name ↑	Model ↑	Network ↑	Type ↑	MAC Address ↑	
nic-0	VirtIO	Pod Networking	masquerade	-	⋮

Add Network Interface

Name *
nic-1

Model *
VirtIO

Network *
host-brl

Type *
bridge

MAC Address

Cancel Add

Next Review and create Back Cancel

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

Project: default ▾

Create Virtual Machine

1 General
2 Networking
3 Storage
4 Advanced
Cloud-init
Virtual Hardware
5 Review
6 Result

Disks Add Disk

Name	Source	Size	Interface	Storage Class
rootdisk	URL	10 GiB	VirtIO	-

1

2

3

4

5

Next Review and create Back Cancel

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

Project: default ▾

Create Virtual Machine

Form Custom script

Hostname

Authenticated SSH Keys

[+ Add SSH Key](#)

Base-64 encoded

Add CD-ROM

Source *
Container

Container *
registry.lab.lan:5000/example-iso-image

Name *
cd-drive-1

Size
Dynamic

Interface *
sata

[Cancel](#) [Add](#)

[Next](#) [Review and create](#) [Back](#) [Cancel](#)

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

The screenshot shows the 'Create Virtual Machine' review page in the Red Hat OpenShift Container Platform. The page is divided into several sections: General, Networking, Storage, and Advanced. The 'Review and confirm your settings' section is highlighted, showing the following configuration:

General

- Name: rhel02
- Description: No description
- Source: URL
- Operating System: Red Hat Enterprise Linux 8.0 or higher
- Flavor: Small: 1 vCPU, 2 GiB Memory
- Workload Profile: desktop

Networking

Name	Model	Network	MAC Address
nic-0	VirtIO	Pod Networking	

Storage

1 ⚠️ Some disks do not have a storage class defined
Default storage class managed-nfs-storage will be used

Name	Source	Size	Interface	Storage Class	Access Mode	Volume Mode
rootdisk	URL	10 GiB	VirtIO		Single User (RWO)	Filesystem

Advanced

- Cloud Init: Not Enabled
- Start virtual machine on creation

2 Create Virtual Machine Back Cancel

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

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

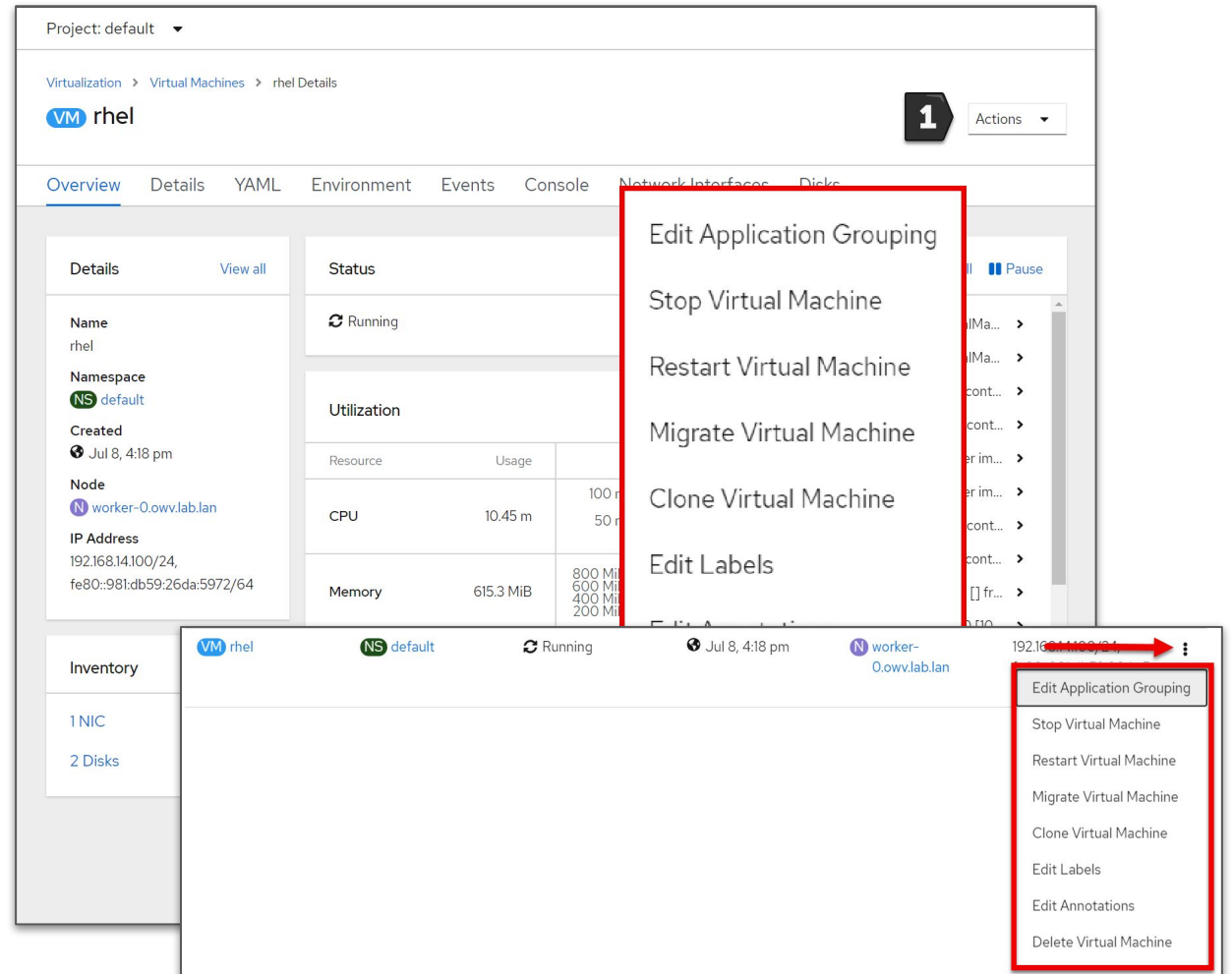
The screenshot displays the OpenShift Virtualization console interface for a virtual machine named 'rhel'. The interface is organized into several sections:

- Details (1):** Located on the left, it provides metadata for the VM, including its name ('rhel'), namespace ('default'), creation time ('Jul 8, 4:18 pm'), node ('worker-0.ovv.lab.lan'), and IP address ('192.168.14.100/24').
- Inventory (2):** Below the details, it shows the hardware configuration, listing '1 NIC' and '2 Disks'.
- Status:** Located at the top right of the main content area, it indicates the VM is 'Running'.
- Utilization (3):** A central section with a '1 Hour' time range selector. It contains a table and three line graphs showing resource usage over time (9:35, 9:40, 9:45).

Resource	Usage	9:35	9:40	9:45
CPU	10.45 m	100 m	50 m	50 m
Memory	615.3 MiB	800 MiB	600 MiB	400 MiB
Filesystem	256.8 MiB	300 MiB	200 MiB	100 MiB
Network Transfer	22.59 KiB in, 11.11 KiB out	30 KiB	20 KiB	10 KiB
- Events:** On the right side, a list of events is shown, including 'VirtualMa...', 'Started cont...', 'Created cont...', 'Container im...', 'Add net1', 'Add eth0', and 'Successfully ...'.

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

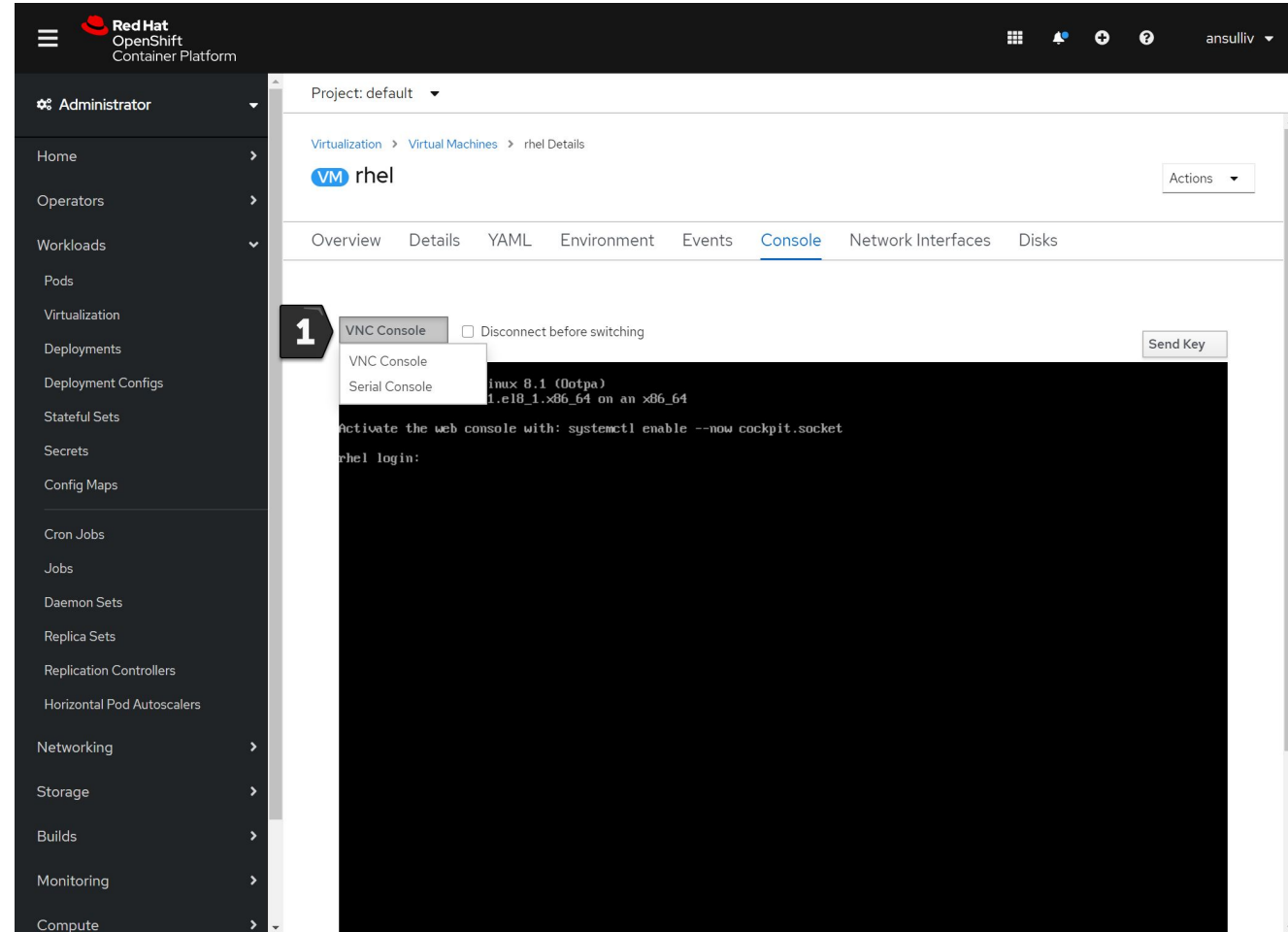
- Details about the virtual machine
 - Labels, annotations
 - Configured OS
 - Template used, if any
 - Configured boot order
 - Associated workload profile
 - Flavor
- Additional details about scheduling
 - Node selector, tolerations, (anti)affinity rules
- Services configured for the VM

The screenshot displays the Red Hat OpenShift Container Platform interface for a Virtual Machine (VM) named 'rhel'. The interface is divided into several sections:

- Virtual Machine Details:**
 - Name:** rhel
 - Status:** Running
 - Namespace:** default
 - Pod:** virt-launcher-rhel-dkdrnd
 - Labels:** app=rhel, flavor.template.kubevirt.io/small=true, os.template.kubevirt.io/rhel8.2=true, vm.kubevirt.io/template=rhel8-desktop-small-v0.10.0, vm.kubevirt.io/template.namespace=openshift, vm.kubevirt.io/template.revision=1, vm.kubevirt.io/template.version=v0.11.2, workload.template.kubevirt.io/desktop=true
 - Annotations:** 3 Annotations
 - Description:** Not available
 - Operating System:** Red Hat Enterprise Linux 8.0 or higher
 - Template:** Not available
 - Created At:** Jul 8, 4:18 pm
 - Owner:** No owner
 - Boot Order:** 1. rootdisk (Disk)
 - CD-ROMs:** Not available
 - IP Address:** 192.168.14.163/24, fe80::87cc:48e1e2:9d23:64
 - Node:** worker-00wvlab.lan
 - Workload Profile:** desktop
- Scheduling and resources requirements:**
 - Node Selector:** No selector
 - Tolerations:** No Tolerations rules
 - Affinity Rules:** No Affinity rules
 - Flavor:** Small: 1 vCPU, 2 GiB Memory
 - Dedicated Resources:** No Dedicated resources applied
- Services:** No Services Found

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

The image displays two screenshots of the Red Hat OpenShift Container Platform console interface, showing the configuration of a virtual machine named 'rhel'.

The top screenshot shows the 'Network Interfaces' tab. It features a table with the following data:

Name	Model	Network	Type	MAC Address
nic-0	VirtIO	host-br1	bridge	-

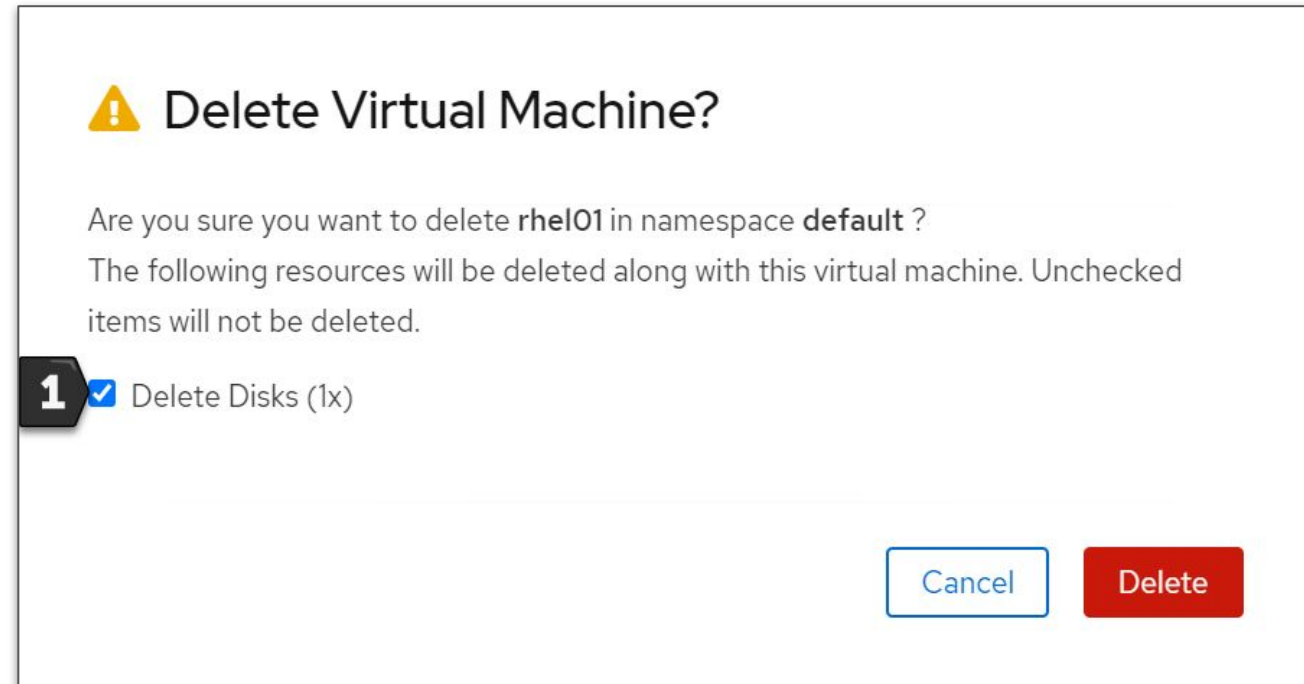
The bottom screenshot shows the 'Disks' tab. It features a table with the following data:

Name	Source	Size	Interface	Storage Class
cloudinitdisk	Other	-	VirtIO	-
rootdisk	URL	20 GiB	VirtIO	managed-nfs-storage

Destroy VMs

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

The image shows two screenshots from the OpenShift console. The top screenshot is the 'rhel' VM overview page, showing a 'Utilization' section with a CPU usage graph. The bottom screenshot is the 'Metrics' panel, which is an enlarged view of the CPU usage graph. The CPU usage graph in the overview shows a value of 36.86 m. The metrics panel shows a line graph of CPU usage over time, with a query editor and a table of results.

Overview Page:

- Project: default
- Virtualization > Virtual Machines > rhel Details
- VM rhel
- Actions
- Overview Details YAML Environment Events Console Network Interfaces Disks
- Status: Running
- Events: View all Pause
- There are no recent events.
- Utilization (1 Hour): CPU 36.86 m

Metrics Panel:

Metrics Prometheus UI

30m Reset Zoom

0.035
0.03
0.025
0.02
0.015
0.01
0.005
0

13:05 13:10 13:15 13:20 13:25 13:30

Insert Metric at Cursor Add Query Run Queries

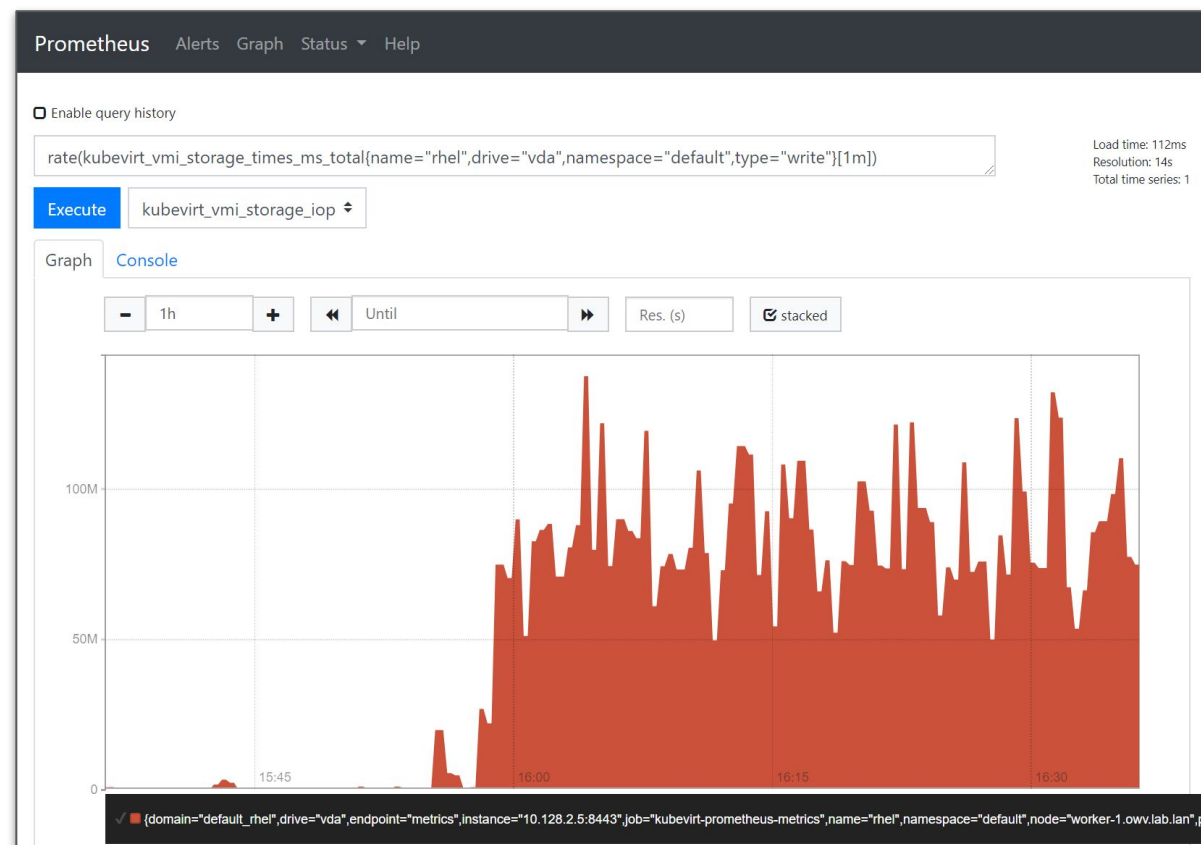
pod:container_cpu_usage:sum{pod='virt-launcher-rhel-24wbf'}

Name	namespace	pod	prometheus	Value
pod:container_cpu_usage:sum	default	virt-launcher-rhel-24wbf	openshift-monitoring/k8s	0.03392109586001908

1-1 of 1 of 1

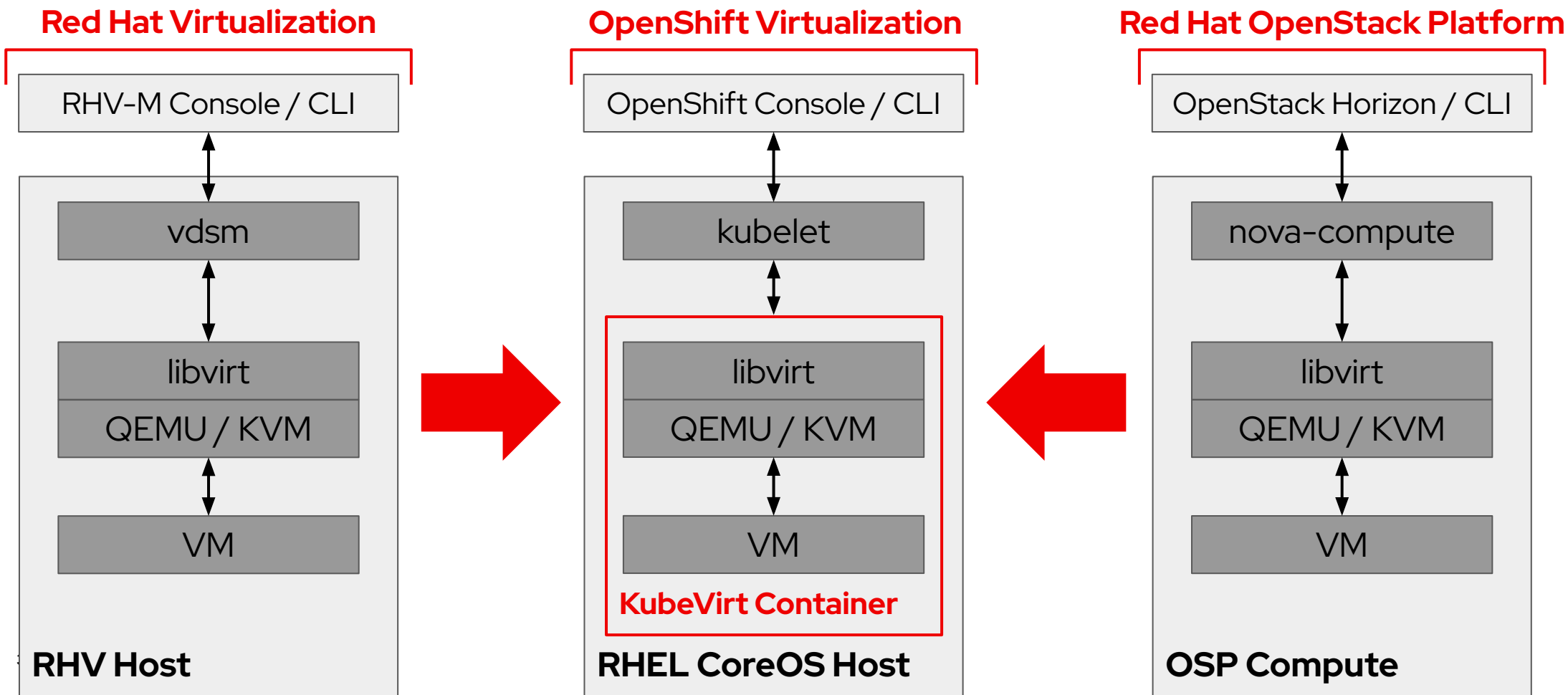
Detailed Virtual Machine metrics

- 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

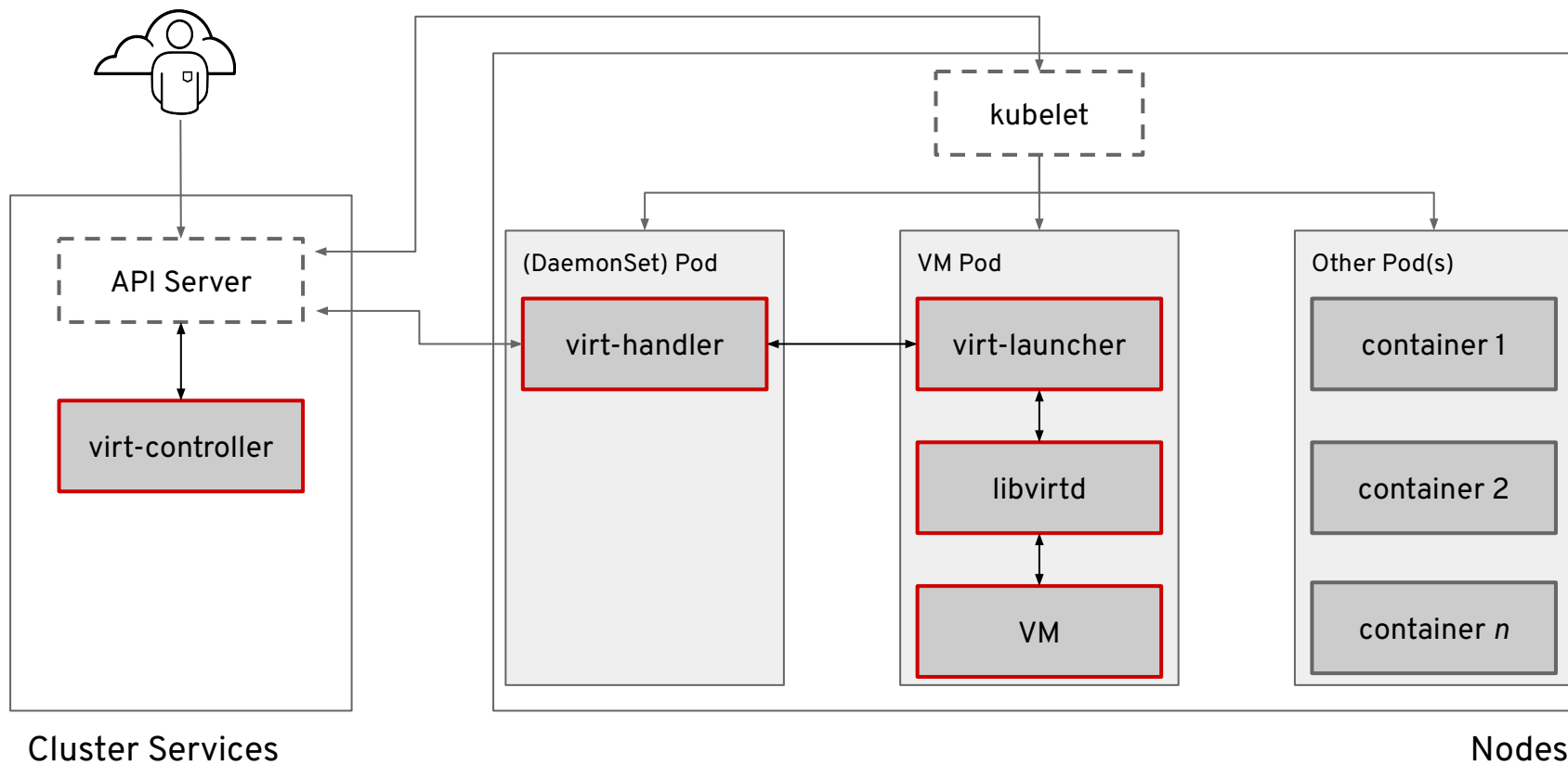


Deeper into the technology

Containerizing KVM



Architectural Overview



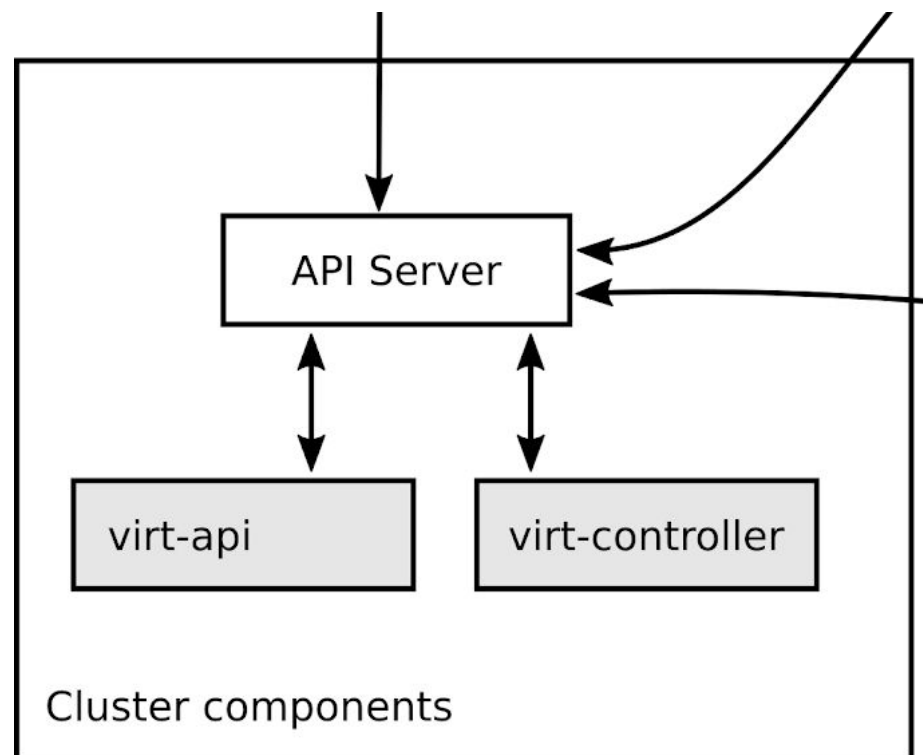
Adding virtualization to the Kubernetes API

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

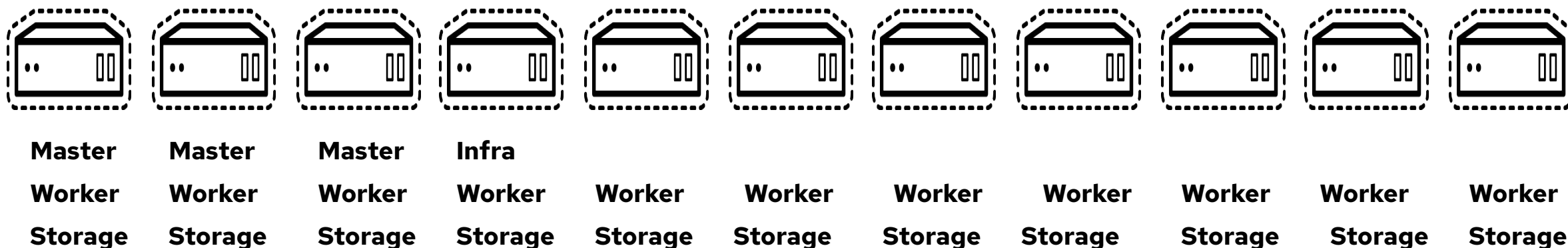
Single API entry point for all workloads

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



Openshift virtualization cluster architecture options

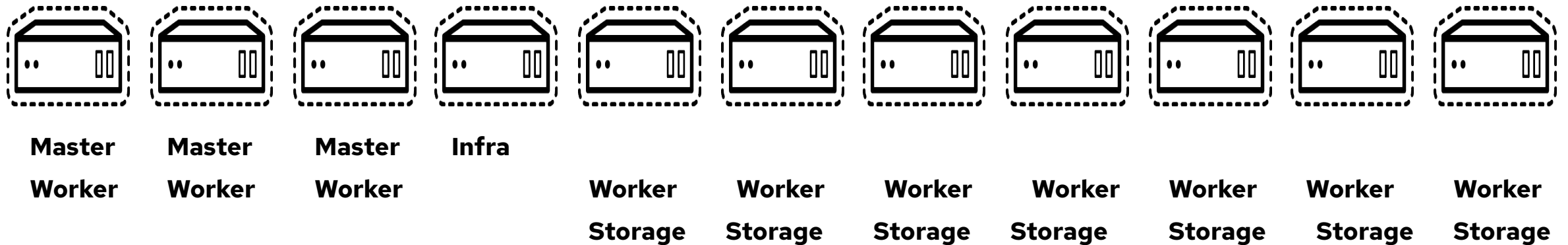
OpenShift cluster architecture #1



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

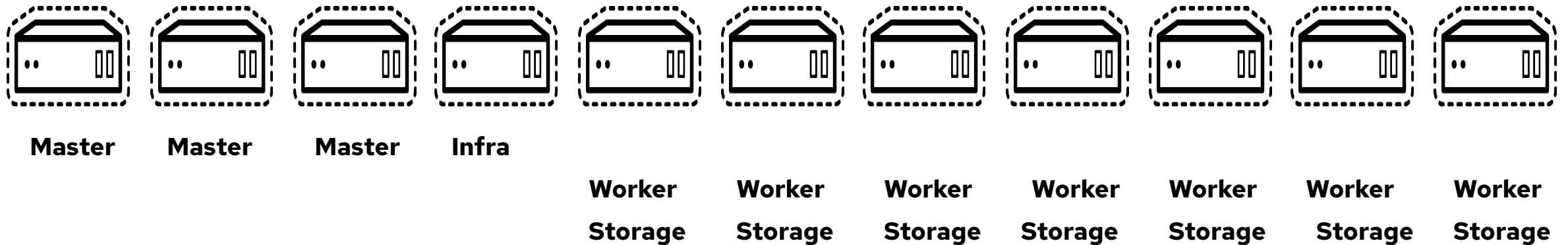
OpenShift cluster architecture #2



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"

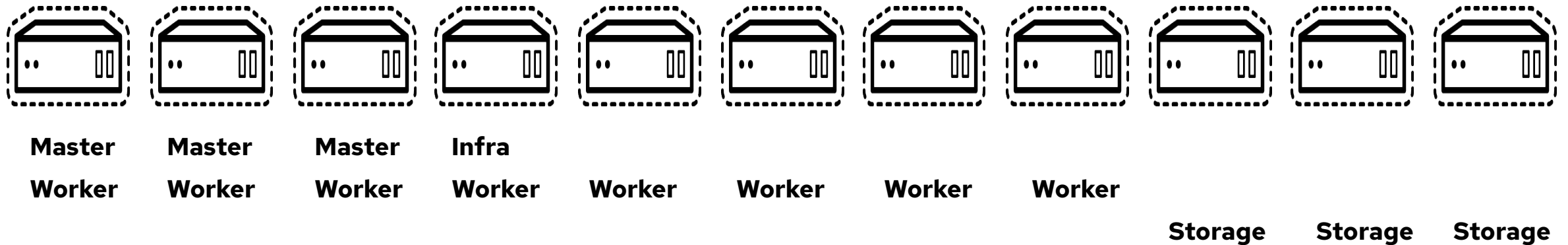
OpenShift cluster architecture #3



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)

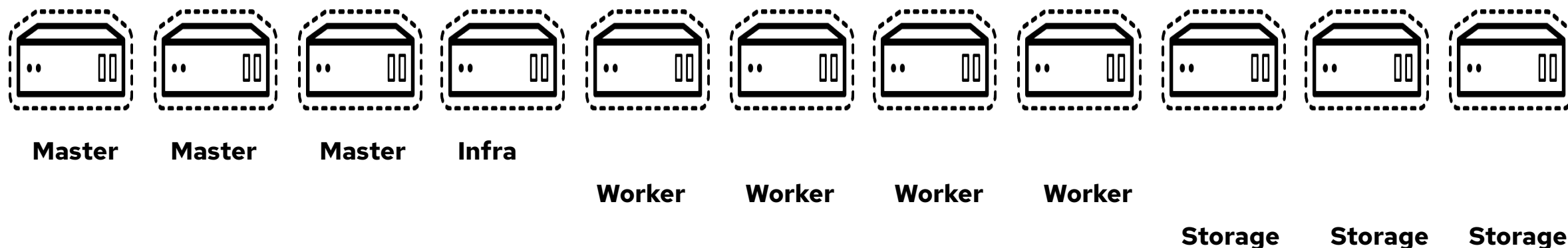
OpenShift cluster architecture #4



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



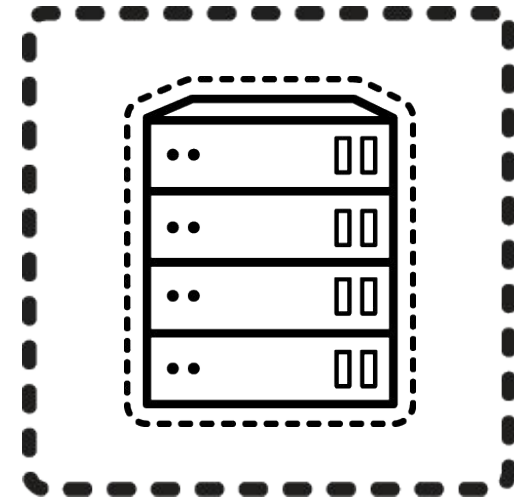
Dedicated everything

- 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

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



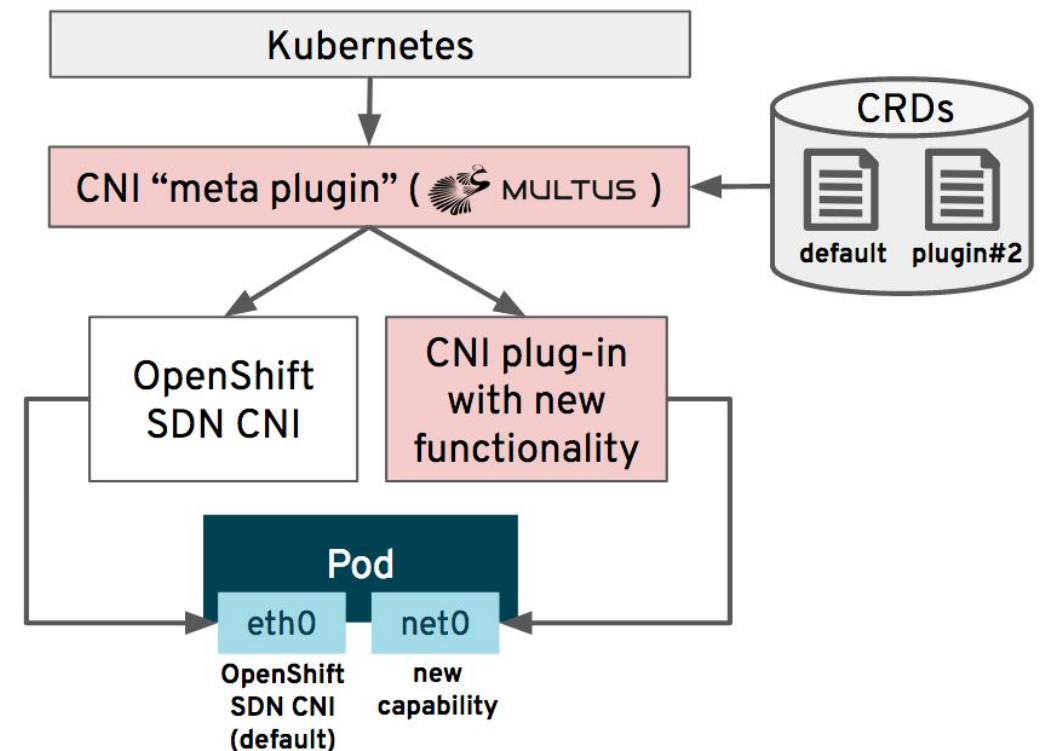
Virtual Machine Instances

- 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

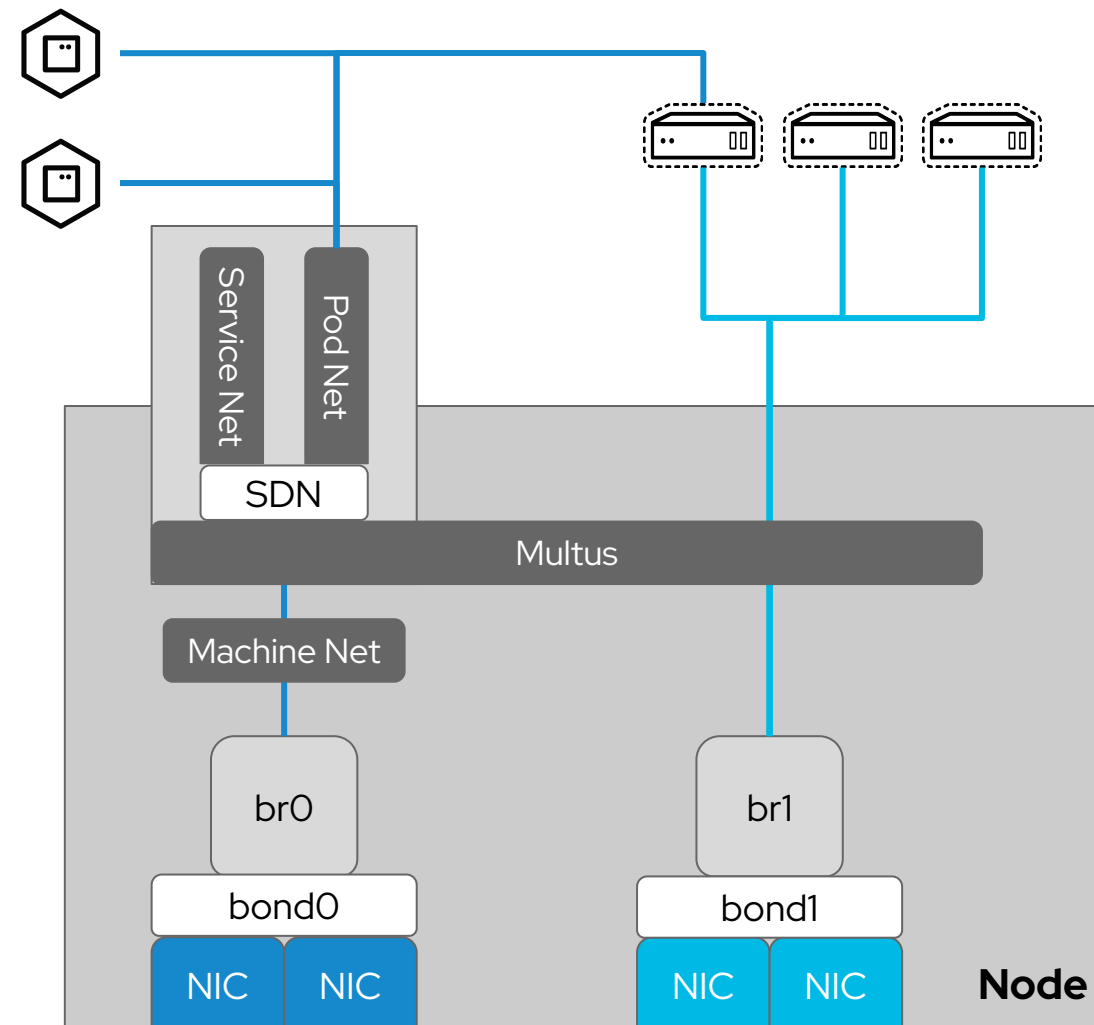
- Virtual machines optionally connect to the standard pod network
 - OpenShift SDN, OVNKubernetes, etc.
- Additional network interfaces accessible via Multus:
 - Bridge, SR-IOV
 - VLAN and other networks can be created using nmstate at the host level
- When using at least one interface on the default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



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

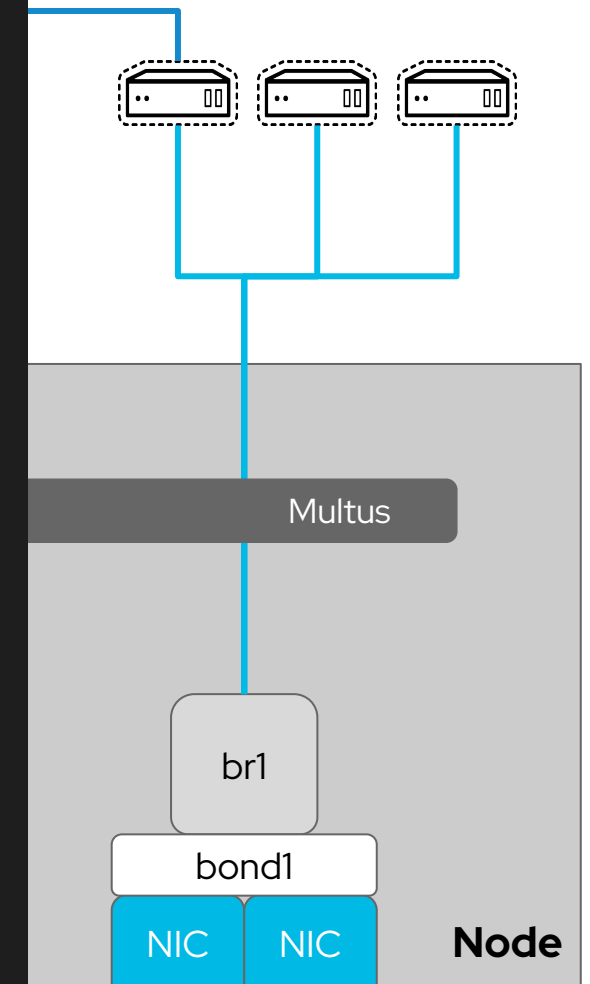
The following slides show an example of how this setup is configured



Host bond configuration

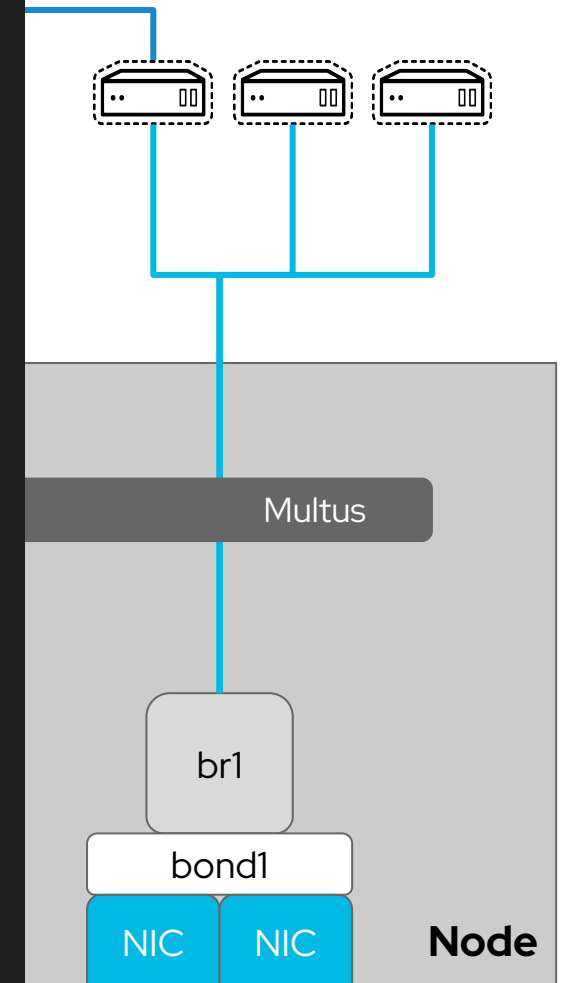
- NodeNetworkConfigurationPolicy (NNCP)
 - Nmstate operator CRD
 - Configure host network using declarative language
- Applies to all nodes specified in the nodeSelector, including newly added nodes automatically
- Update or add new NNCPs for additional host configs

```
1  apiVersion: nmstate.io/v1alpha1
2  kind: NodeNetworkConfigurationPolicy
3  metadata:
4    name: worker-bond1
5  spec:
6    nodeSelector:
7      node-role.kubernetes.io/worker: ""
8    desiredState:
9      interfaces:
10     - name: bond1
11       type: bond
12       state: up
13       ipv4:
14         enabled: false
15       link-aggregation:
16         mode: balance-alb
17         options:
18           miimon: '100'
19         slaves:
20           - eth2
21           - eth3
22       mtu: 1450
```



Host bridge configuration

```
1  apiVersion: nmstate.io/v1alpha1
2  kind: NodeNetworkConfigurationPolicy
3  metadata:
4    name: worker-bond1-br1
5  spec:
6    nodeSelector:
7      node-role.kubernetes.io/worker: ""
8    desiredState:
9      interfaces:
10     - name: br1
11       description: br1 with bond1
12       type: linux-bridge
13       state: up
14       ipv4:
15         enabled: false
16       bridge:
17         options:
18           stp:
19             enabled: false
20         port:
21     - 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`

```
1  API Version:  nmstate.io/v1alpha1
2  Kind:         NodeNetworkConfigurationEnactment
3  Name:        worker-1.ovv.lab.lan.worker-br1-bond1
4  Status:
5    Conditions:
6      Last Heartbeat Time:  2020-07-08T20:15:46Z
7      Last Transition Time: 2020-07-08T20:15:46Z
8      Message:              successfully reconciled
9      Reason:               SuccessfullyConfigured
10     Status:               True
11     Type:                 Available
12   Desired State:
13     Interfaces:
14       Bridge:
15         Options:
16           Stp:
17             Enabled: false
18         Port:
19           Name:      bond1
20         Description: br1 with bond1
21     ipv4:
22       Enabled: false
23       Name:    br1
24       State:  up
25       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

```
1  apiVersion: k8s.cni.cncf.io/v1
2  kind: NetworkAttachmentDefinition
3  metadata:
4    name: br1-public
5    annotations:
6      k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
7  spec:
8    config: '{
9      "cniVersion": "0.3.1",
10     "name": "br1-public",
11     "plugins": [
12       {
13         "type": "cnv-bridge",
14         "bridge": "br1"
15       },
16       {
17         "type": "cnv-tuning"
18       }
19     ]
20   }'
```

```
1  kind: Pod
2  apiVersion: v1
3  metadata:
4    name: application-pod
5    annotations:
6      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

```
1  apiVersion: kubevirt.io/v1alpha3
2  kind: VirtualMachine
3  |   name: demo-vm
4  spec:
5  |   template:
6  |     |   spec:
7  |     |     |   domain:
8  |     |     |     |   devices:
9  |     |     |     |     |   interfaces:
10 |     |     |     |     |     |   - bridge: {}
11 |     |     |     |     |     |     |   model: virtio
12 |     |     |     |     |     |     |   name: nic-0
13 |     |     |     |     |   hostname: demo-vm
14 |     |     |     |   networks:
15 |     |     |     |     |   - multus:
16 |     |     |     |     |     |   networkName: bond1-br1
17 |     |     |     |     |     |   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

PersistentVolumeClaim Details

Name	rhel-rootdisk	Status	✔ Bound
Namespace	NS default	Capacity	20Gi
Labels	app=containerized-data-importer	Access Modes	ReadWriteMany
Annotations	12 Annotations ✎	Volume Mode	Filesystem
Label Selector	No selector	Storage Class	SC managed-nfs-storage
Created At	🕒 Jul 8, 4:18 pm	Persistent Volume	PV pvc-alaac411-2e46-495a-897e-cf3bc2442199
Owner	DV rhel-rootdisk		

VM disks in PVCs

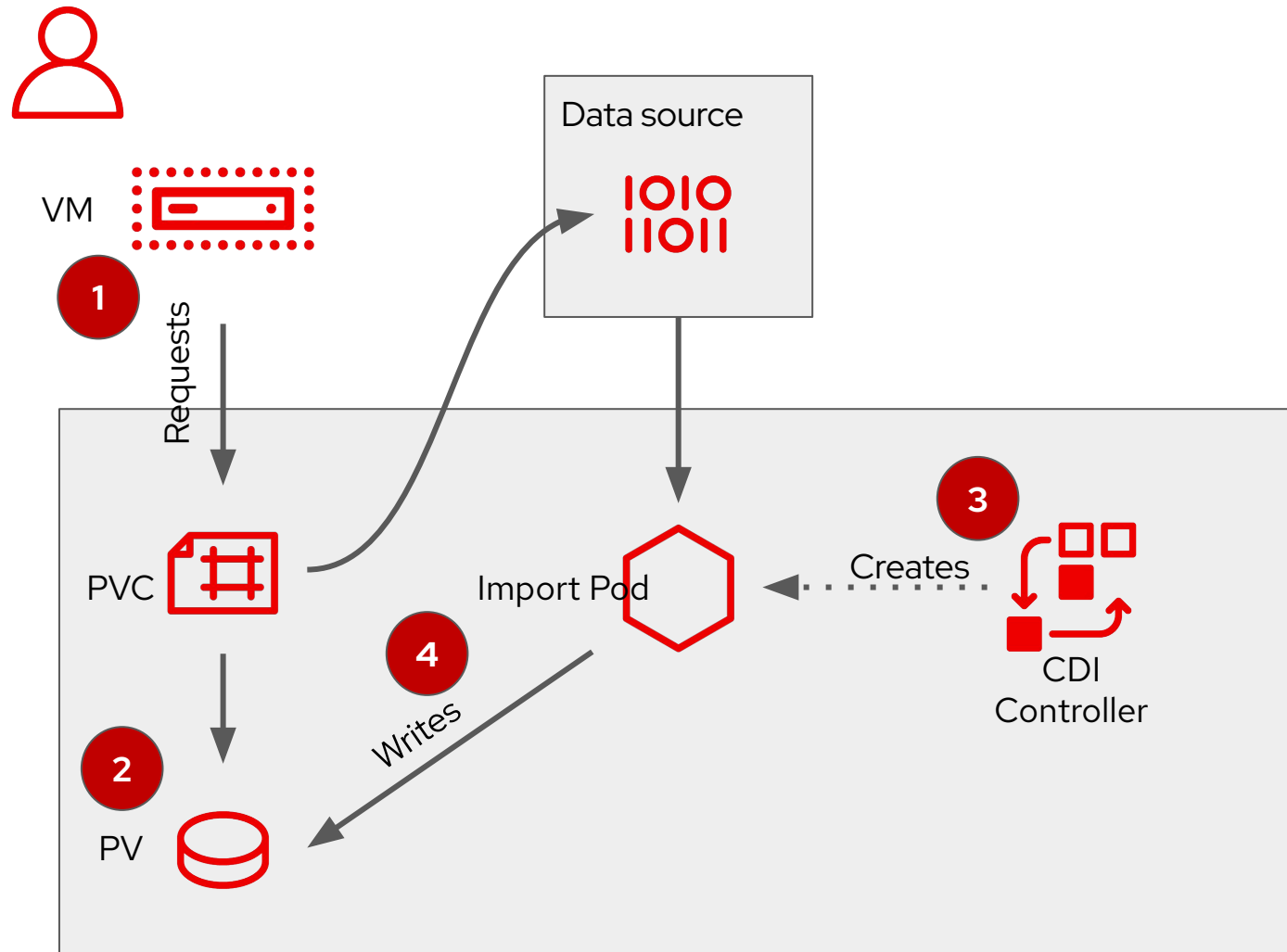
- VM disks on FileSystem PVCs are created as thin provisioned raw images
 - Thick provisioned disks are not created by CDI, may be possible manually
- Block PVCs are attached directly to the VM
- CSI operations, e.g. snapshot and clone, are not supported with VM disk PVCs
 - Use DataVolumes to clone 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 via 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

```
1  dataVolumeTemplates:
2    - apiVersion: cdi.kubevirt.io/v1alpha1
3      kind: DataVolume
4      metadata:
5        creationTimestamp: null
6        name: vm-rootdisk
7      spec:
8        pvc:
9          accessModes:
10         - ReadWriteMany
11         resources:
12           requests:
13             storage: 20Gi
14           storageClassName: my-storage-class
15           volumeMode: Filesystem
16        source:
17          http:
18            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

```
1 spec:
2   domain:
3     disks:
4       - bootOrder: 1
5         disk:
6           bus: virtio
7           name: rootdisk
8     volumes:
9       - containerDisk:
10         image: registry.lab.lan:5000/fedora:31
11         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

```
1 spec:
2   domain:
3     devices:
4       - disk:
5         bus: virtio
6         name: cloudinitdisk
7     volumes:
8       - cloudInitNoCloud:
9         userData: |-
10            #cloud-config
11            password: redhat
12            chpasswd: { expire: False }
13            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

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

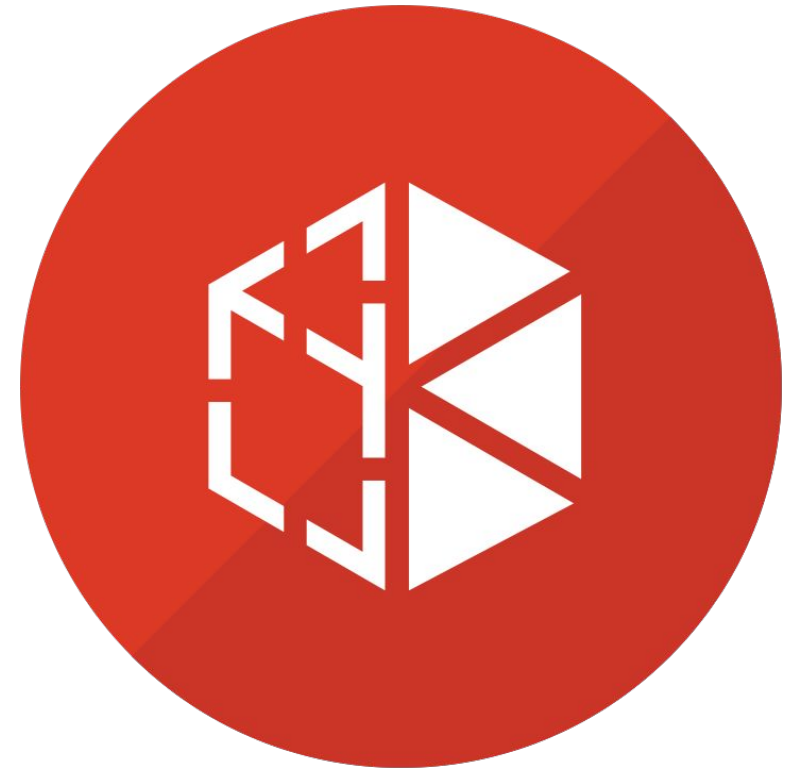
V0000000



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](https://www.linkedin.com/company/red-hat)

 [youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

 [facebook.com/redhatinc](https://www.facebook.com/redhatinc)

 twitter.com/RedHat