



# OpenShift Virtualization

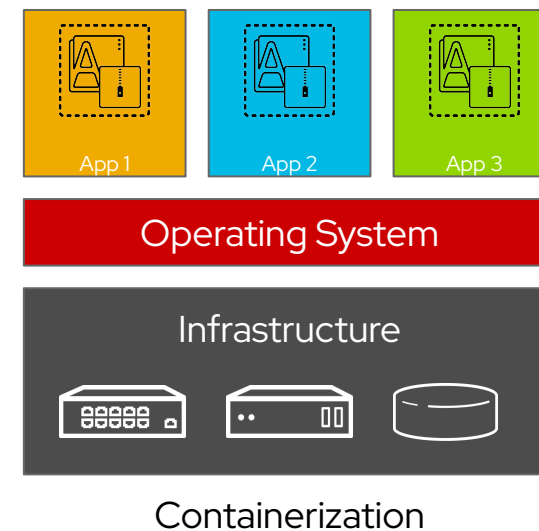
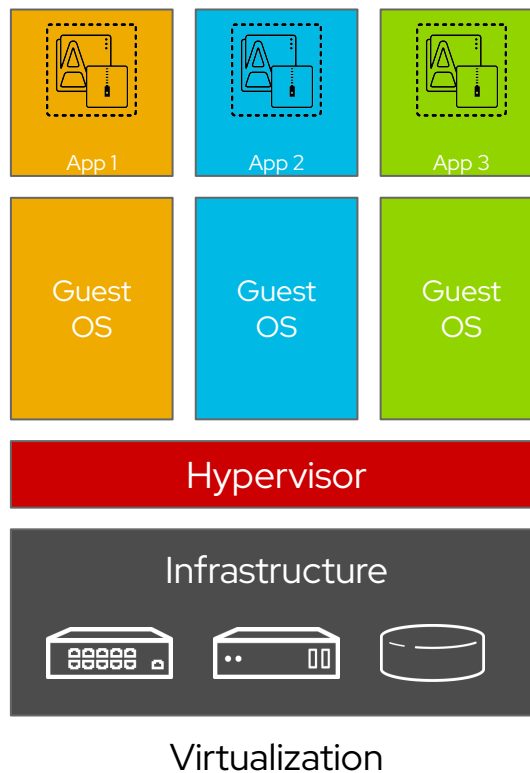
## Technical Deep Dive

Alfred Bach  
PSA Red Hat  
Field Partner and Learning Team

# What is OpenShift Virtualization?

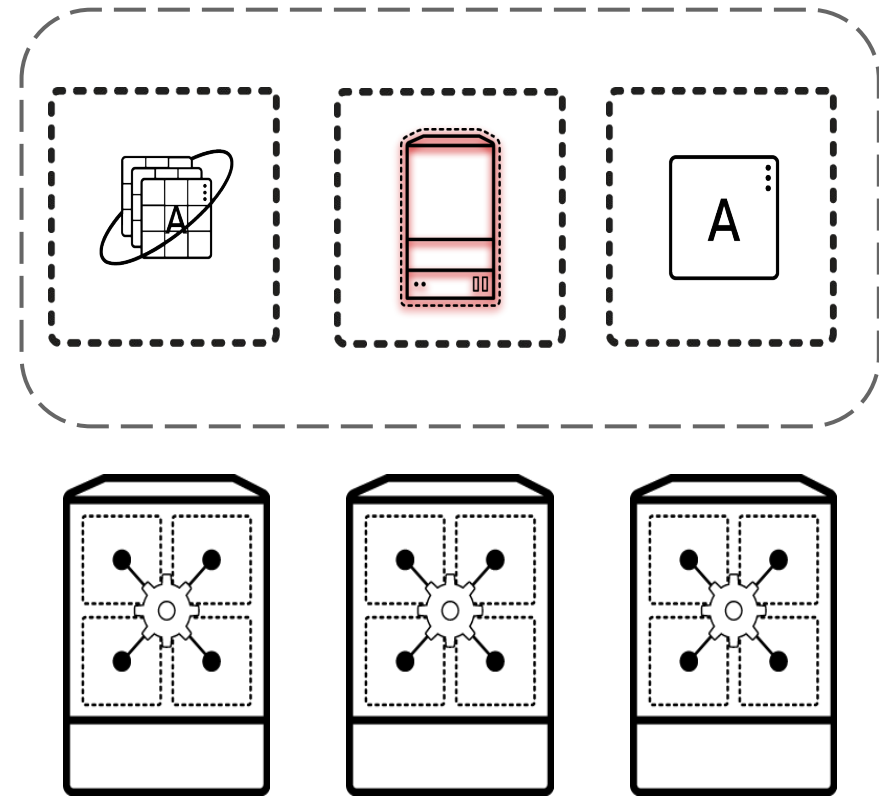
# Containers are not virtual machines

- 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



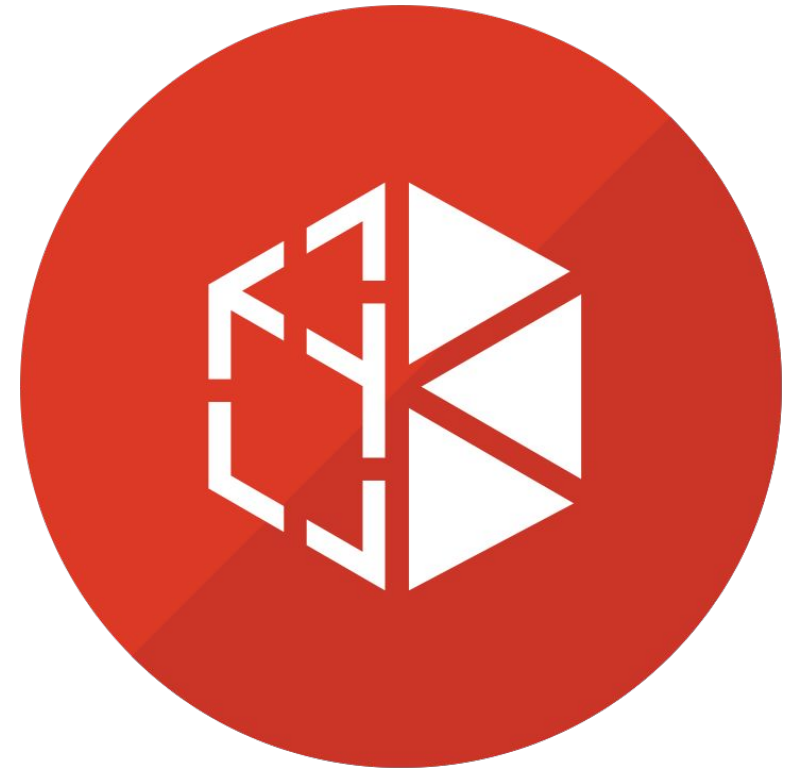
# Virtual machines can be put into containers

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



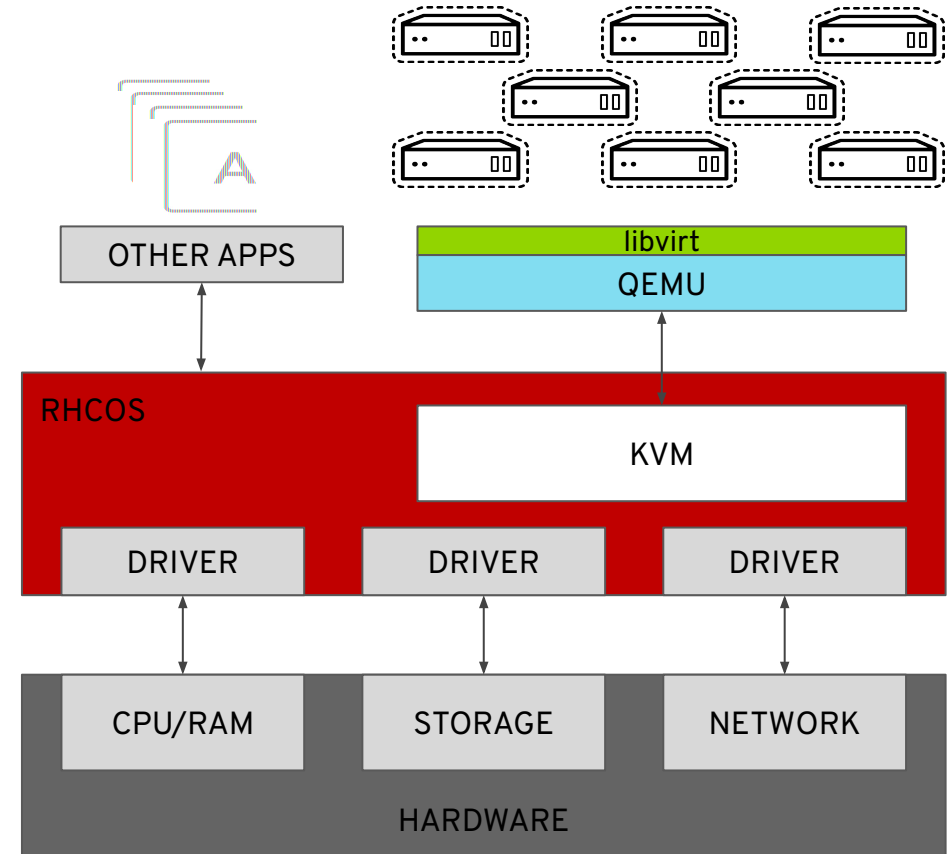
# OpenShift Virtualization

- Virtual machines
  - Running in containers, managed as Pods
  - 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)



# OpenShift Virtualization uses KVM

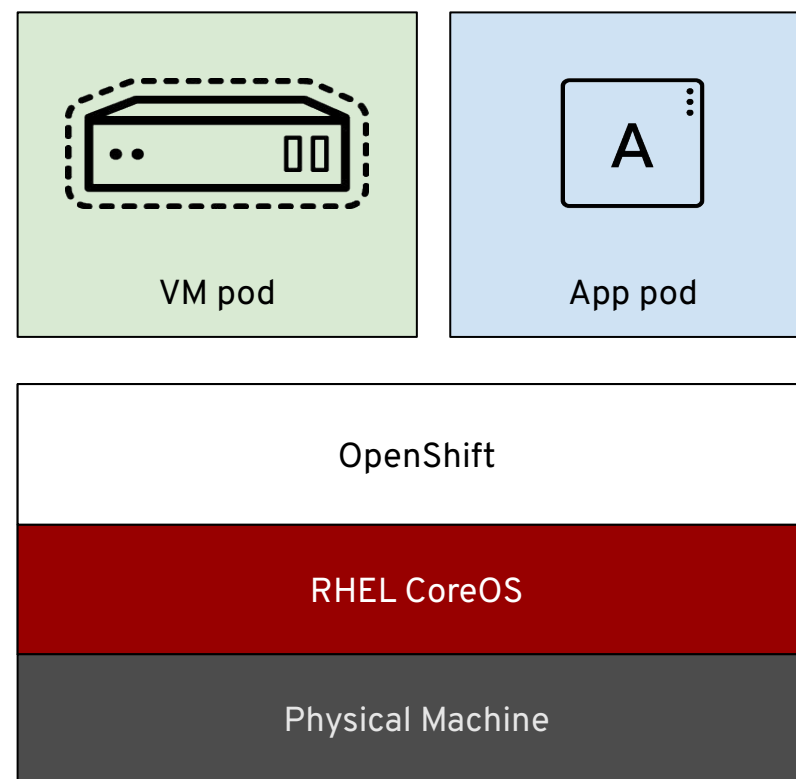
- OpenShift Virtualization uses KVM, the Linux kernel hypervisor
- KVM is a core component of the Red Hat Enterprise Linux kernel
  - KVM has 10+ years of production use: Red Hat Virtualization, Red Hat OpenStack Platform, and RHEL all leverage KVM, QEMU, and libvirt
- QEMU uses KVM to execute virtual machines
- **libvirt** provides a management abstraction layer
- Currently supported on x86 bare metal
- For other platforms contact Product Management for roadmap



# 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



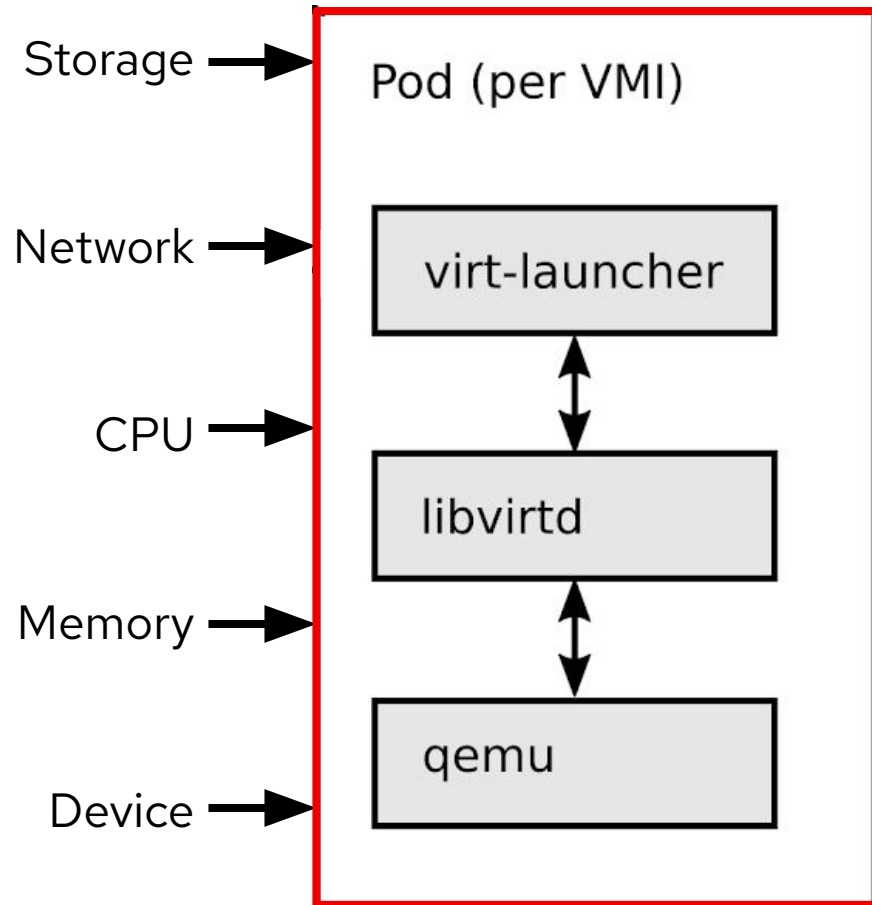


# Virtualization native to Kubernetes

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

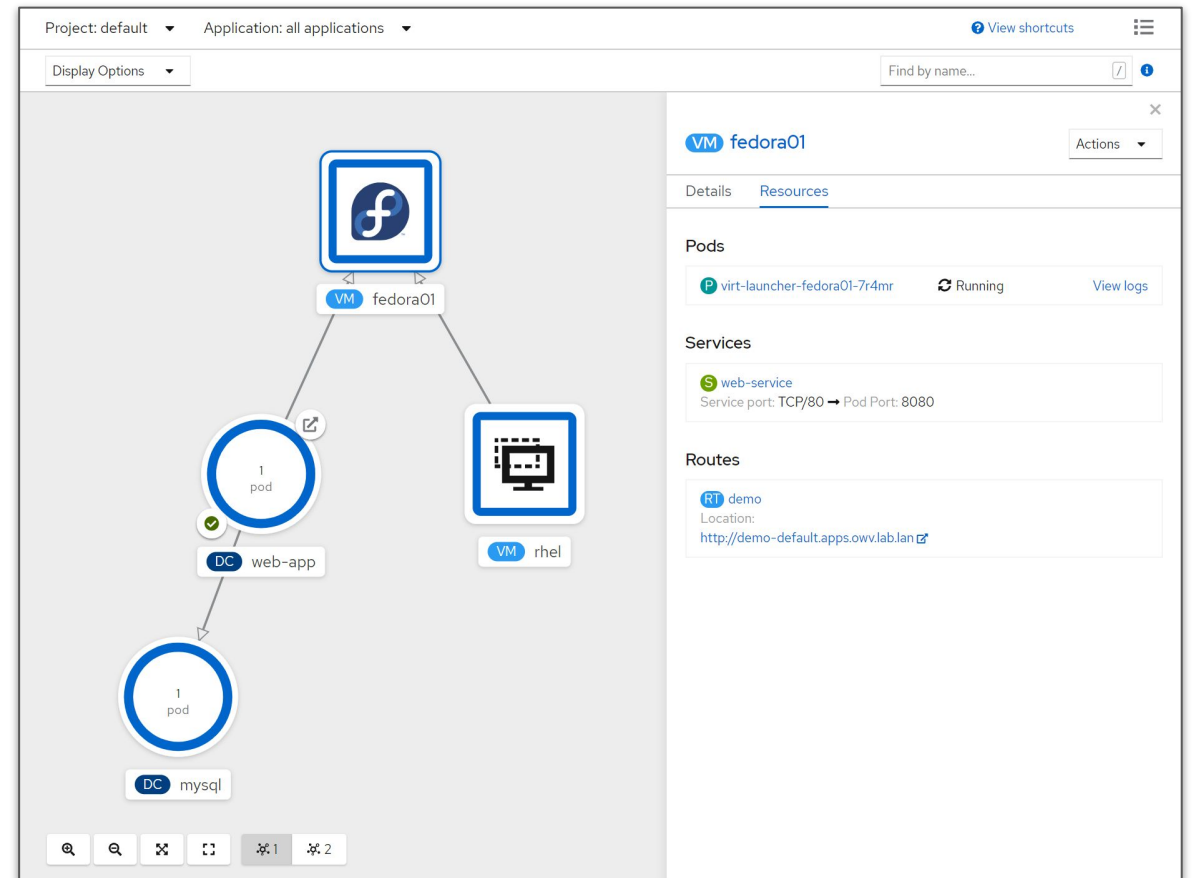
- libvirt and qemu from RHEL are mature, have high performance, provide stable abstractions, and have a minimal overhead.

## Security - Defense in depth

- RHCOS has controlled configuration by default, SELinux MCS, plus KVM isolation - inherited from the Red Hat portfolio stack

# Using VMs and containers together

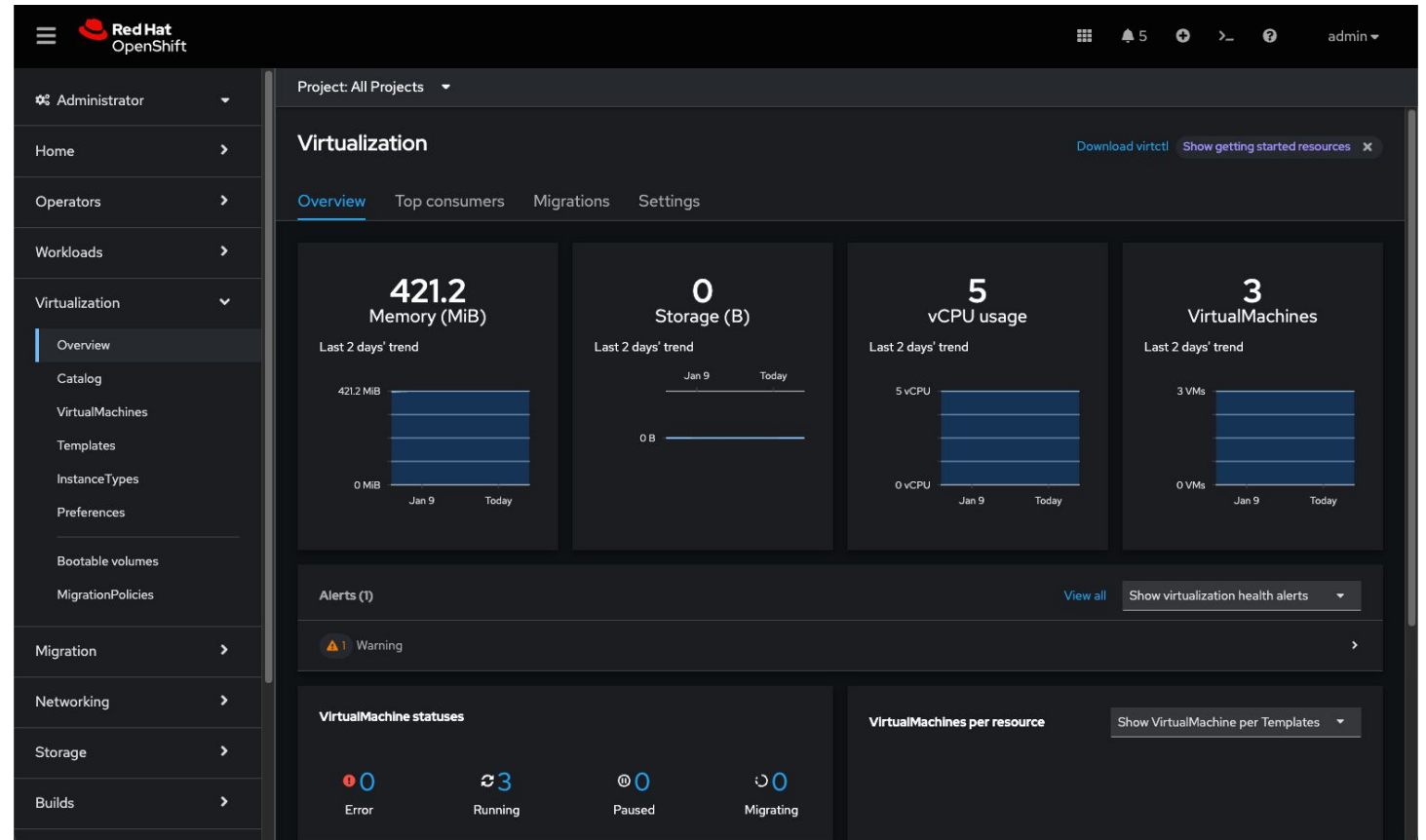
- Virtual machines connected to SDN networks are accessible using standard Kubernetes methods:
  - Service, Route, Ingress
  - Service Mesh
  - Pipelines
- Network policies apply to VM pods the same as application pods
- VM-to-Pod, and vice-versa, communication happens over SDN or ingress depending on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



# Managed with OpenShift

# Virtual Machine Management

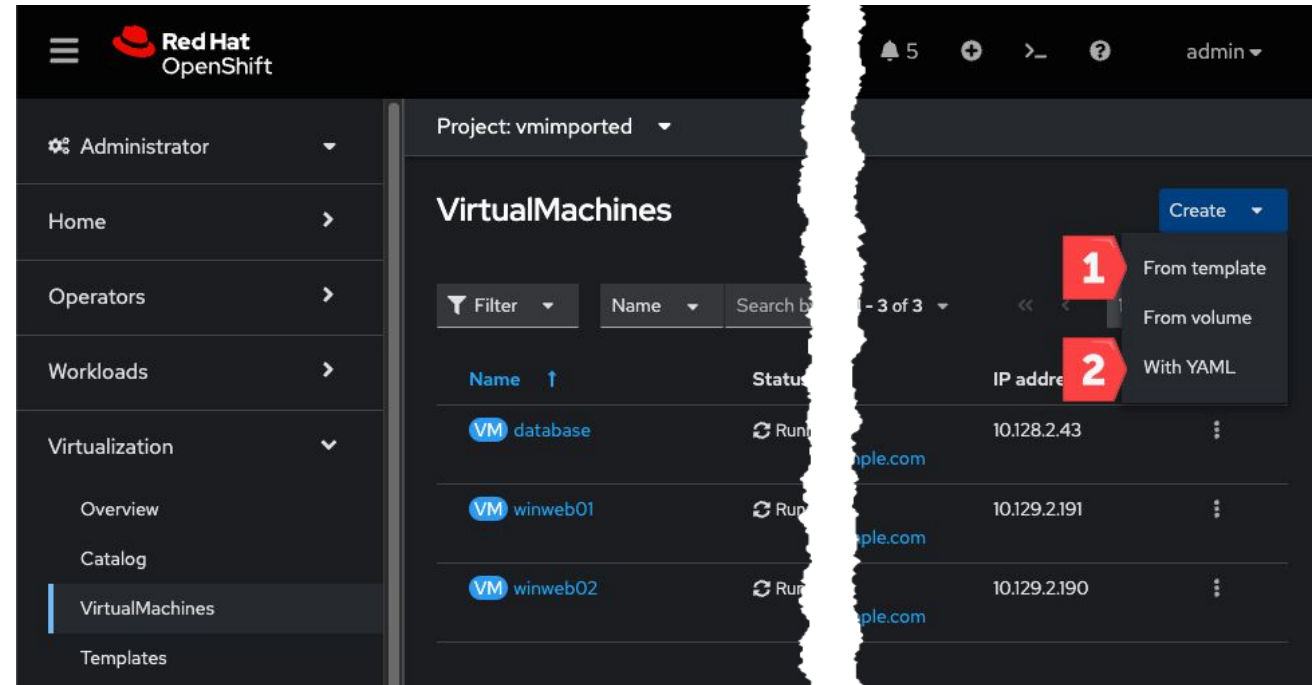
- Create, modify, and destroy virtual machines, and their resources, using the OpenShift web interface or CLI
- Use the `virtctl` command to simplify virtual machine interaction from the CLI



# Create VMs

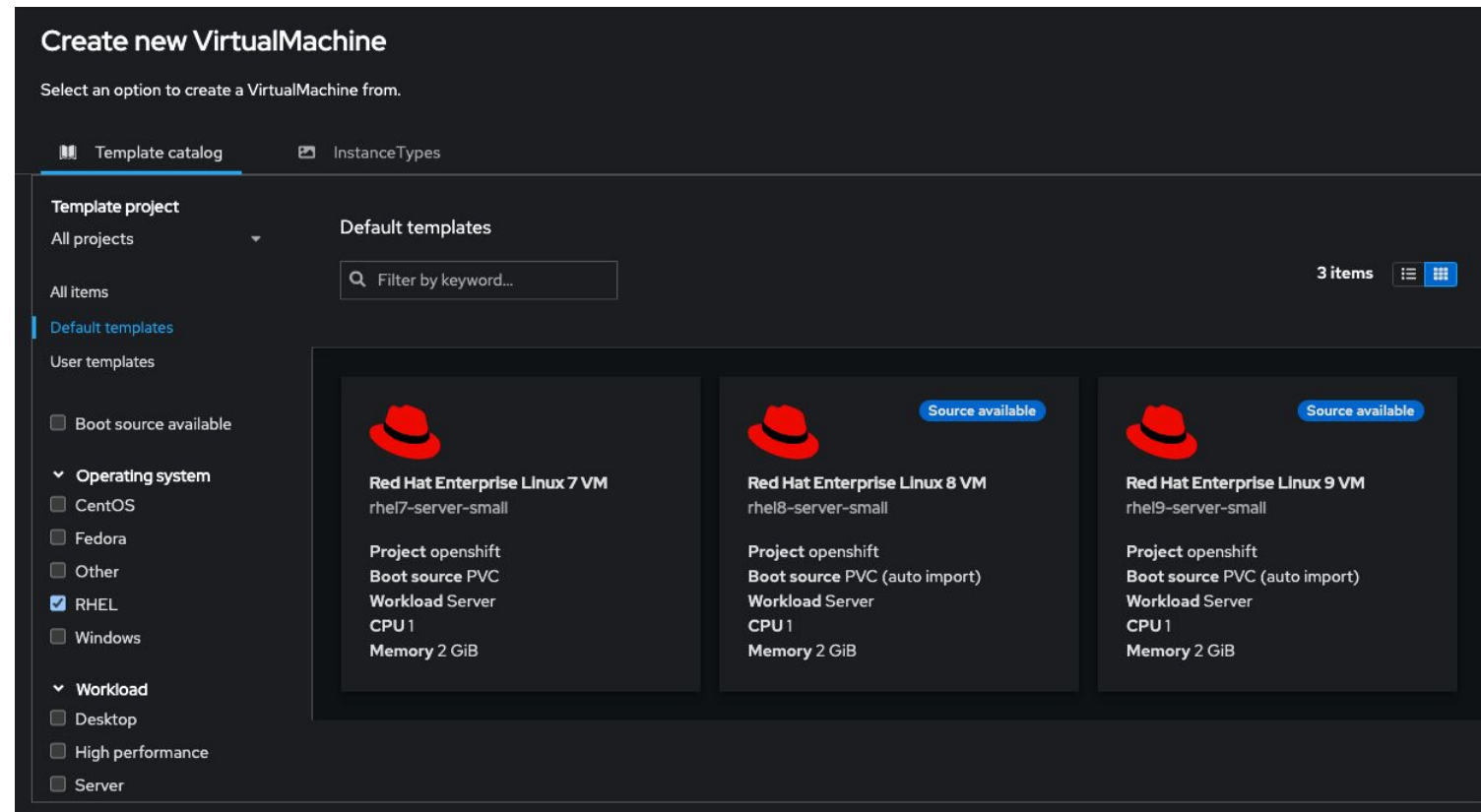
# Virtual Machine creation

- 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



# Using templates for virtual machines


- Simplified and streamlined virtual machine creation experience for VM consumers
- Administrators configure templates with an OS disk, consumers select from the options





# Creating a virtual machine

- Pre-defined, but customizable, CPU and memory
- A default network configuration is defined in the template, for example using the SDN or an external network
- Storage, including PVC size and storage class, are determined by the template administrator
- Guest OS customization, e.g. cloud-init, is inherited from the template
- The VM can be further customized by selecting the option, or immediately created and deployed
  - Additional customization includes CPU/memory, storage, network, cloud-init, and more

 **Red Hat Enterprise Linux 8 VM**  
rhel8-server-small

**Template info**

**Operating system**  
Red Hat Enterprise Linux 8 VM

**Workload type**  
Server (default)

**Description**  
Template for Red Hat Enterprise Linux 8 VM or newer. A PVC with the RHEL disk image must be available.

**Documentation**  
[Refer to documentation](#)

**1 CPU | Memory**  
1 CPU | 2 GiB Memory

**2 Network interfaces (1)**

Name	Network	Type
default	Pod networking	Masquerade

**3 Disks (2)**

Name	Drive	Size
rootdisk	Disk	30 GiB
cloudinitdisk	Disk	-

**Hardware devices (0)**  
**GPU devices**  
Not available

**Host devices**  
Not available

**Quick create VirtualMachine**

**VirtualMachine name \***  
rhel8-revolutionary-antelope

**Project**  
vmimported

☒ Start this VirtualMachine after creation

**4 Quick create VirtualMachine** **Customize VirtualMachine** **Cancel**

# Templates

- Fully customizable template options simplify the creation of VMs
- Red Hat provides default templates, administrators can create and customize additional as needed
- Boot sources are populated by the Operator by default. Admins create / import additional as needed

Project: All Projects ▾

## VirtualMachine Templates

Supported operating systems are labeled below. [Learn more about Red Hat support](#)

Filter ▾ Name ▾ rhel9-server

Name rhel9-server ✕ Clear all filters

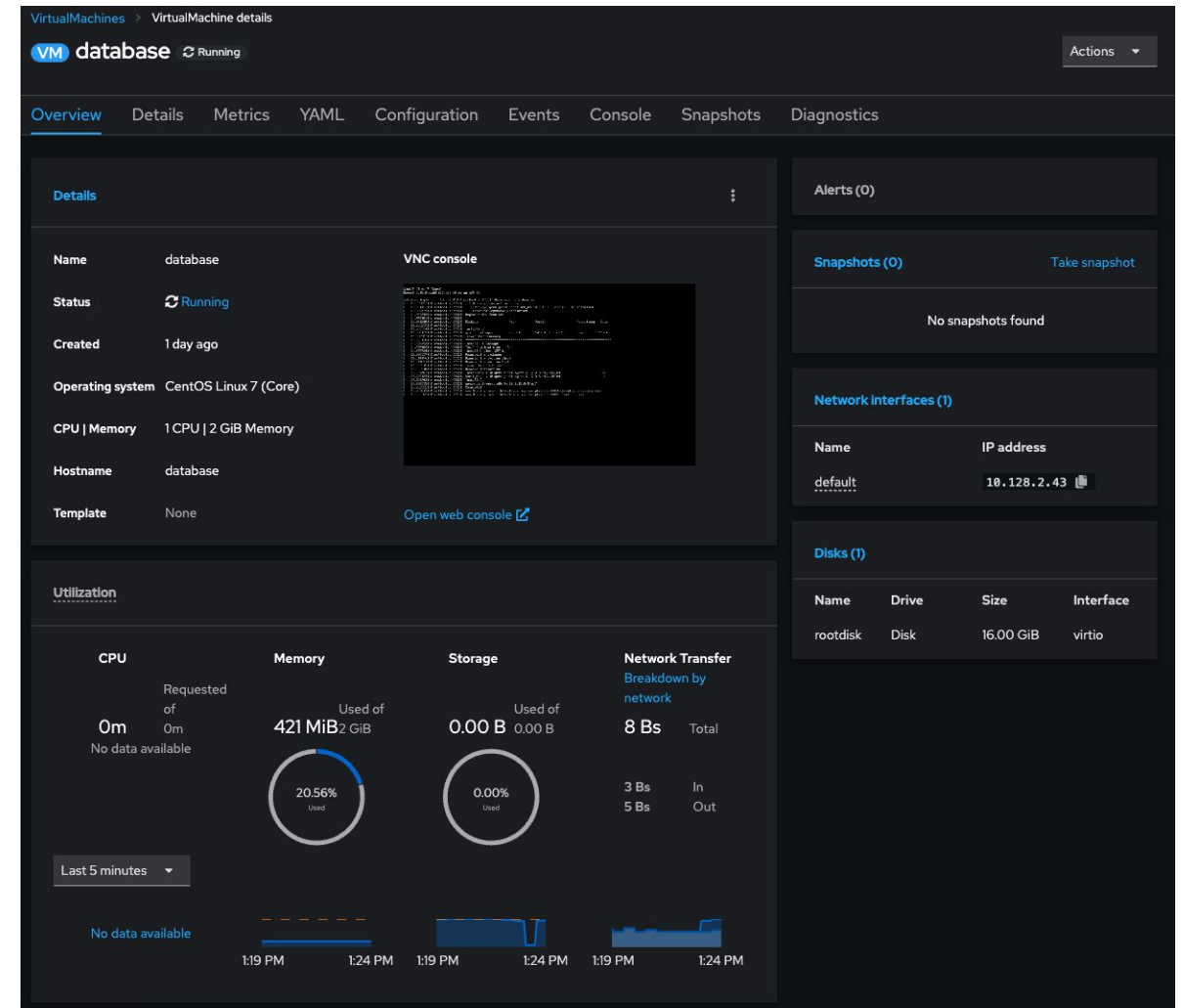
Name ↑	Workload profile	Boot source
rhel9-server-large	Server	PVC (auto import) <a href="#">Source available</a>
rhel9-server-medium	Server	PVC (auto import) <a href="#">Source available</a>
rhel9-server-small	Server	PVC (auto import) <a href="#">Source available</a>
rhel9-server-tiny	Server	PVC (auto import) <a href="#">Source available</a>

2 [Create Template](#)

# View / manage VMs

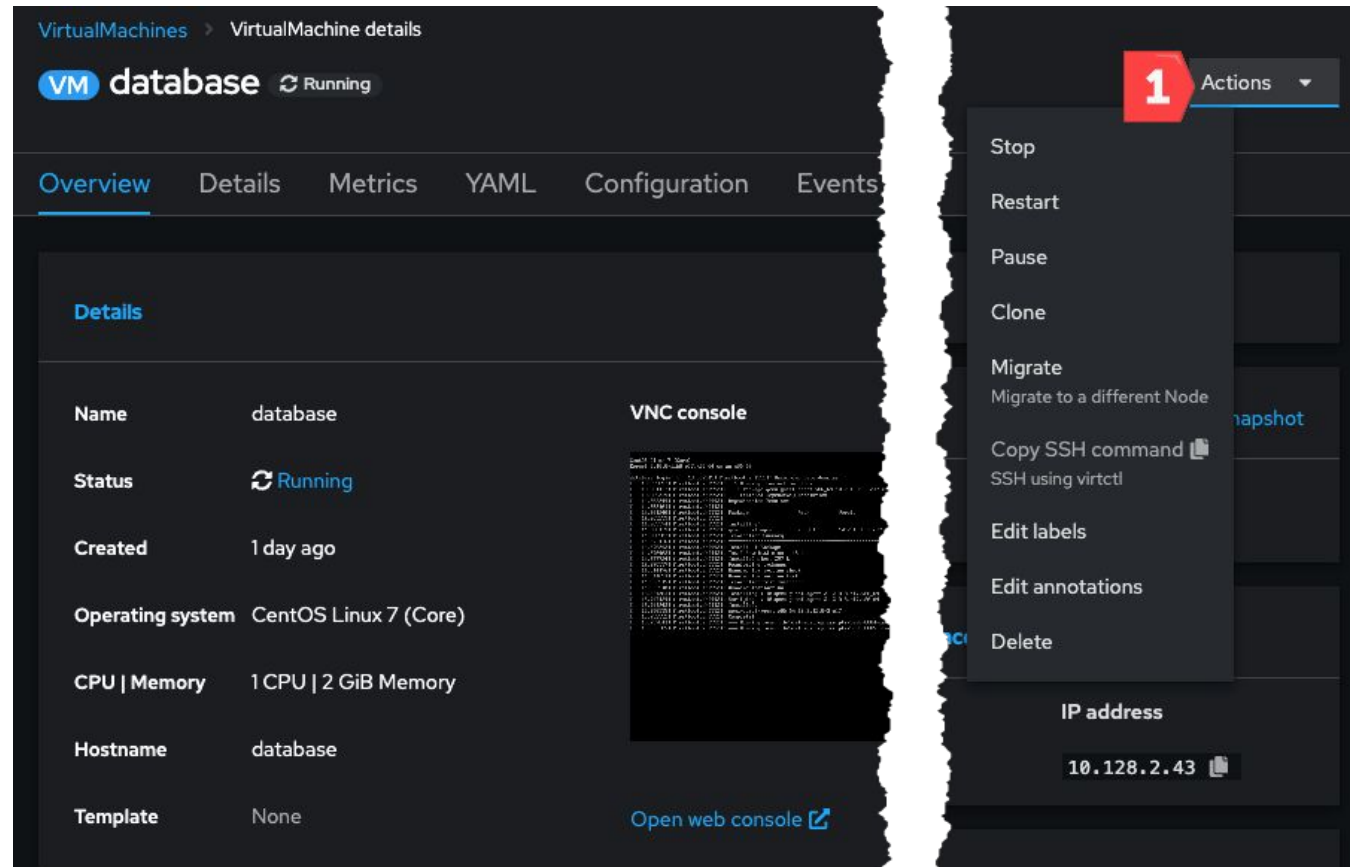
# Virtual Machine – Overview

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



# Virtual Machine – Actions

- Control the state and status of the virtual machine
- 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 primary 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 shows the 'VirtualMachine details' page for a VM named 'database'. The page is divided into two columns. The left column contains details about the VM's configuration, including its name, namespace, labels, annotations, description, operating system, CPU and memory resources, machine type, boot mode, and start in pause mode. The right column contains details about the VM's status and configuration, including its status, pod, virtual machine instance, boot order, IP address, hostname, time zone, node, workload profile, SSH access, and SSH service type. The VM is currently running and has a status of 'Running'.

Property	Value
Name	database
Status	Running
Namespace	NS vmimported
Pod	virt-launcher-database-r2pqz
Labels	app=database
VirtualMachineInstance	database
Annotations	4 Annotations
Description	None
Operating system	CentOS Linux 7 (Core)
CPU   Memory	1 CPU   2 GiB Memory
Machine type	pc-q35-rhel9.2.0
Boot order	1. rootdisk 2. default
IP address	10.128.2.43
Hostname	database
Time zone	EST
Node	ocp4-worker2.aio.example.com
Workload profile	Server
SSH access	Linux only
SSH using virtctl	SSH secret not configured
SSH service type	None
Start in pause mode	Off
Template	None
Created at	

# 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`

The screenshot shows the OpenShift VM console interface. At the top, there's a navigation bar with tabs: Overview, Details, Metrics, YAML, Configuration, Events, **Console** (highlighted with a red '1'), Snapshots, and Diagnostics. Below the tabs, there's a 'Console' section. On the left, there's a dropdown menu for 'Guest login credentials' (highlighted with a red '2') and a 'Send key' dropdown (highlighted with a red '3'). To the right of these is a 'Paste' button and a 'Disconnect' button. The main area displays the console output for a CentOS Linux 7 (Core) VM. The output shows the installation of the `qemu-guest-agent` package, including dependency resolution, transaction check, and installation progress. The output is as follows:

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64

database login: [ 13.557945] firstboot.sh[992]: Resolving Dependencies
[ 13.559697] firstboot.sh[992]: --> Running transaction check
[ 13.561137] firstboot.sh[992]: ---> Package qemu-guest-agent.x86_64 10:2.12.0-3.el7 will be installed
[ 13.969239] firstboot.sh[992]: --> Finished Dependency Resolution
[ 14.253249] firstboot.sh[992]: Dependencies Resolved
[ 14.258961] firstboot.sh[992]: =====
[ 14.261248] firstboot.sh[992]: Package           Arch      Version           Repository    Size
[ 14.263299] firstboot.sh[992]: =====
[ 14.265374] firstboot.sh[992]: Installing:
[ 14.266429] firstboot.sh[992]: qemu-guest-agent      x86_64     10:2.12.0-3.el7   base         116 k
[ 14.268507] firstboot.sh[992]: Transaction Summary
[ 14.273136] firstboot.sh[992]: =====
[ 14.275292] firstboot.sh[992]: Install 1 Package
[ 14.276463] firstboot.sh[992]: Total download size: 116 k
[ 14.277724] firstboot.sh[992]: Installed size: 297 k
[ 14.278937] firstboot.sh[992]: Downloading packages:
[ 14.514176] firstboot.sh[992]: Running transaction check
[ 14.544691] firstboot.sh[992]: Running transaction test
[ 14.832515] firstboot.sh[992]: Transaction test succeeded
[ 14.833940] firstboot.sh[992]: Running transaction
[ 15.534263] firstboot.sh[992]: Installing : 10:qemu-guest-agent-2.12.0-3.el7.x86_64
[ 15.747139] firstboot.sh[992]: Verifying : 10:qemu-guest-agent-2.12.0-3.el7.x86_64
[ 15.749343] firstboot.sh[992]: Installed:
[ 15.758395] firstboot.sh[992]: qemu-guest-agent.x86_64 10:2.12.0-3.el7
[ 15.752952] firstboot.sh[992]: Complete!
[ 15.797041] firstboot.sh[992]: === Running /usr/lib/virt-sysprep/scripts/5000-0004-setenforce-restore ===
[ 15.822305] firstboot.sh[992]: === Running /usr/lib/virt-sysprep/scripts/5000-0005-start-qga ===
```

# Virtual Machine Configuration – Disks and NICs

- Add, edit, and remove NICs and disks for virtual machines using the Configuration tab
- For guests with the qemu guest agent installed, configuration and status reporting is displayed for the relevant objects

VirtualMachines > VirtualMachine details

VM database Running

Overview Details Metrics YAML Configuration Events Console Snapshots Diagnostics

### Disks

1 Add disk

Filter Search by name... Mount Windows drivers disk

Name	Source	Size	Drive	Interface	Storage class
rootdisk bootable	PVC database	16.00 GiB	Disk	virtio	ocs-storagecluster-ceph-rbd

### File systems

Name	File system type	Mount point	Total bytes	Used bytes
dm-0	xfs	/	0.00	0.00
vda1	xfs	/boot	0.00	0.00

VirtualMachines > VirtualMachine details

VM database Running

Overview Details Metrics YAML Configuration Events Console Snapshots Diagnostics

### Network interfaces

1 Add network interface

Filter Name Search by name...

Name	Model	Network	Type	MAC address
default	virtio	Pod networking	Masquerade	02:5c:80:00:00:00



# Virtual Machine Diagnostics

- Consolidated location for status and configuration affecting features of the virtual machine
- Quickly determine the guest status and whether disks are snapshot capable

The screenshot displays the 'VirtualMachine details' page for a VM named 'database' which is in a 'Running' state. The 'Diagnostics' tab is selected, showing two main sections: 'Status conditions' and 'Volume snapshot status'.

**Status conditions**

Type	Status	Reason	Message
Ready	True	-	-
LiveMigratable	True	-	-
AgentConnected	True	-	-

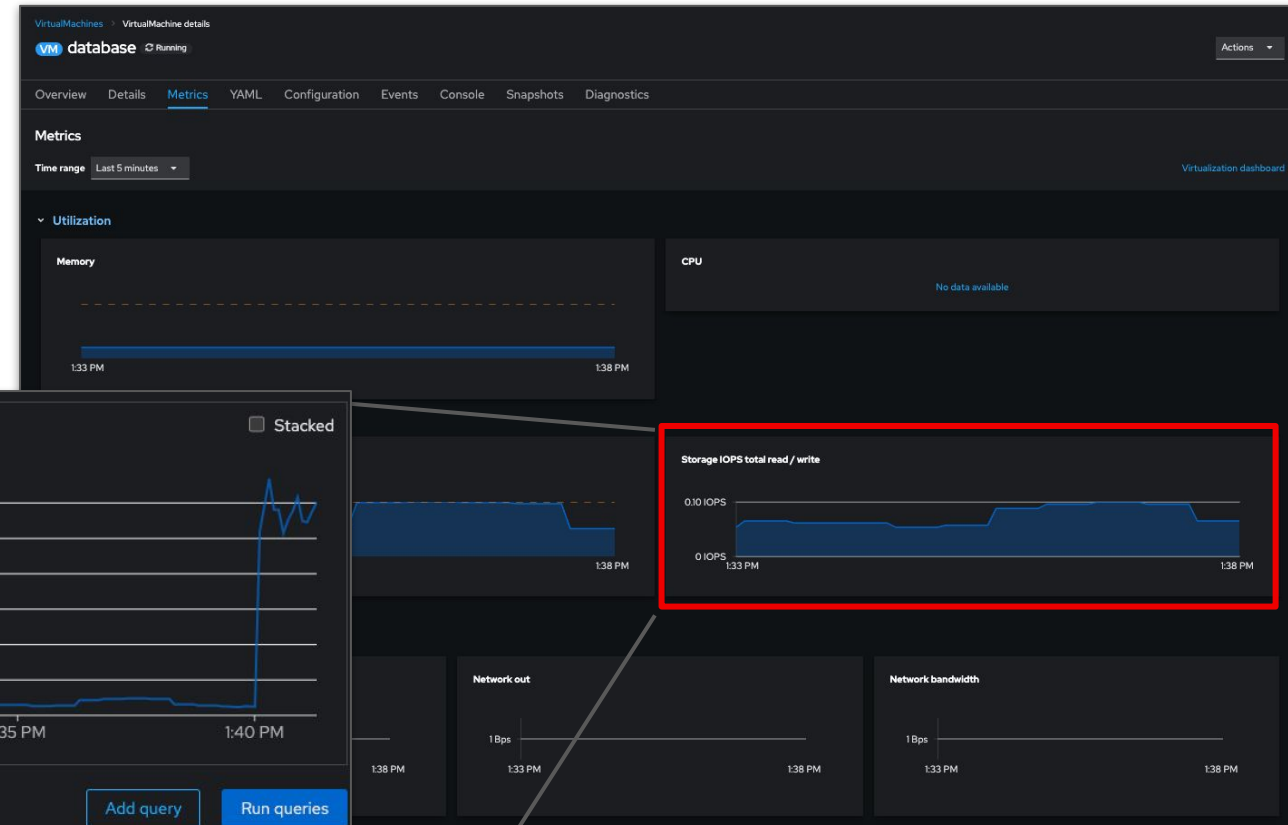
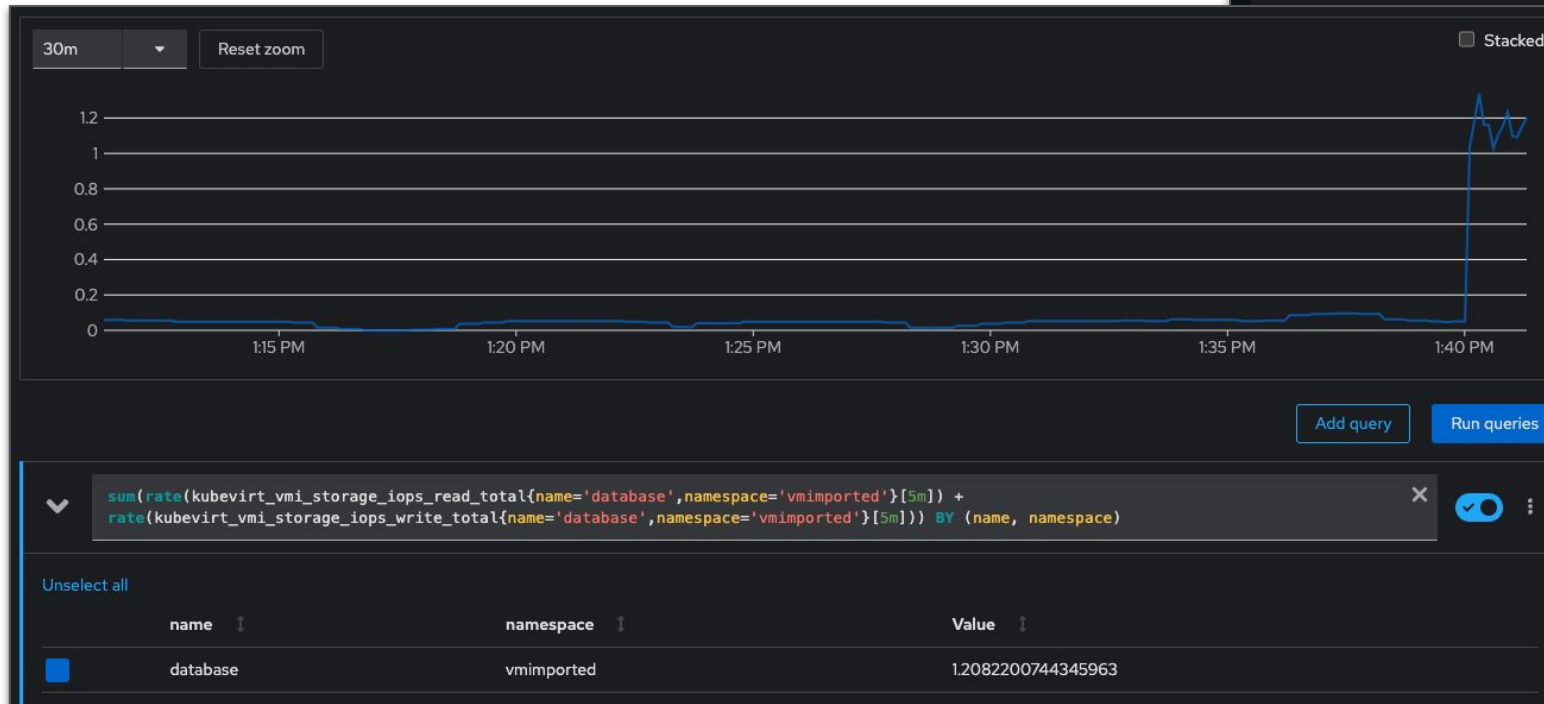
**Volume snapshot status**

Name	Enabled	Reason
rootdisk	true	rootdisk

# Metrics

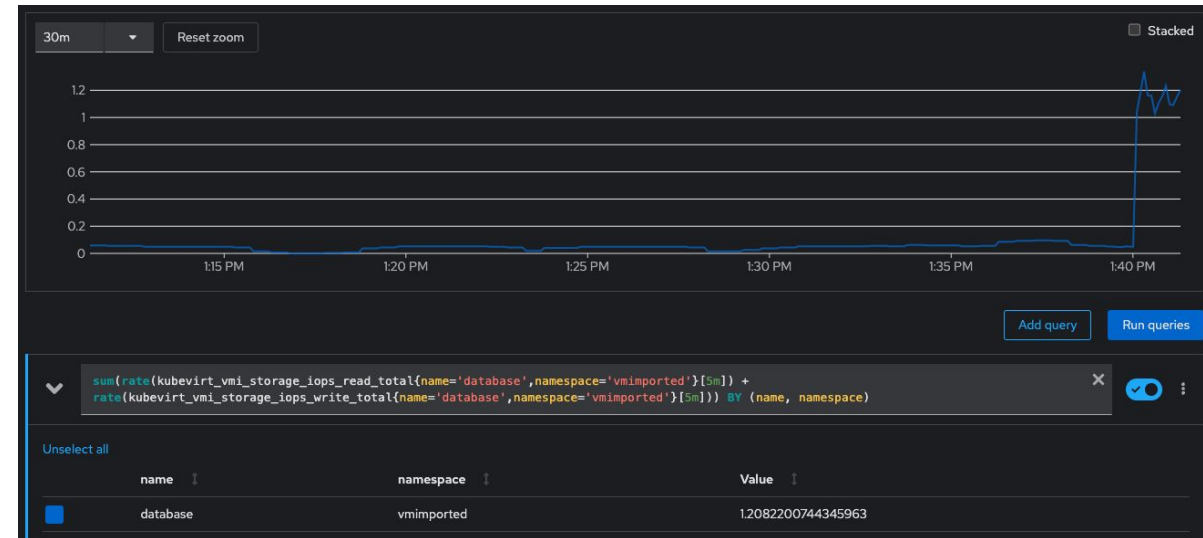
# Overview Virtual Machine metrics

- Virtual machine information tabs contain overview metrics
  - CPU, memory
  - Storage (IOPS, R/W), network (in/out)
  - Migrations



# Detailed Virtual Machine metrics

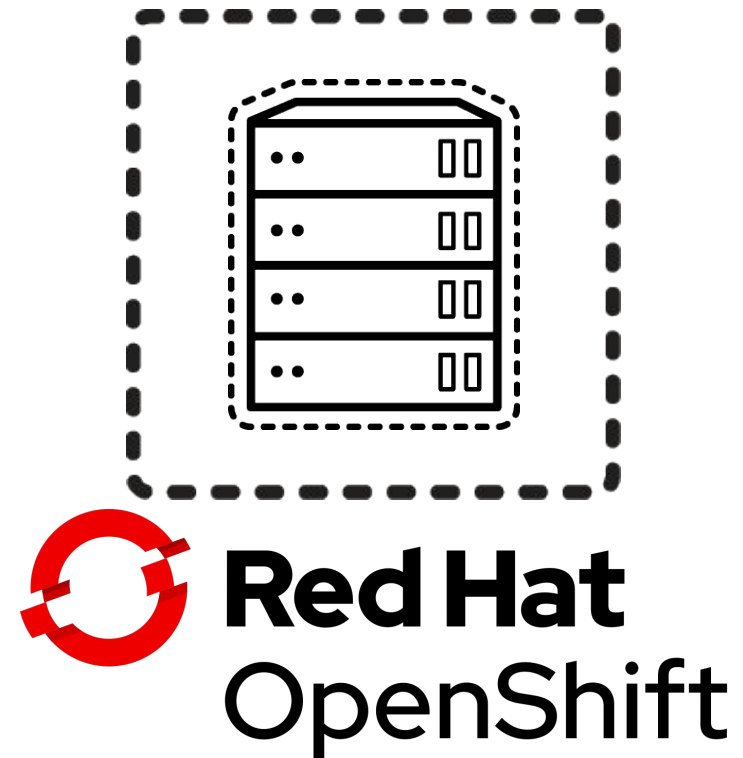
- Virtual machine, and VM pod, metrics are collected by the OpenShift metrics service
- Available per-VM metrics include, but are not limited to:
  - Active memory
  - Active CPU time
  - Network in/out errors, packets, and bytes
  - Storage R/W IOPS, latency, and throughput
- VM metrics are for VMs, not for VM pods
  - Management overhead not included in output
- Create custom dashboards in the same way as with other Pods and applications



# Virtual machines

# Containerized virtual machines

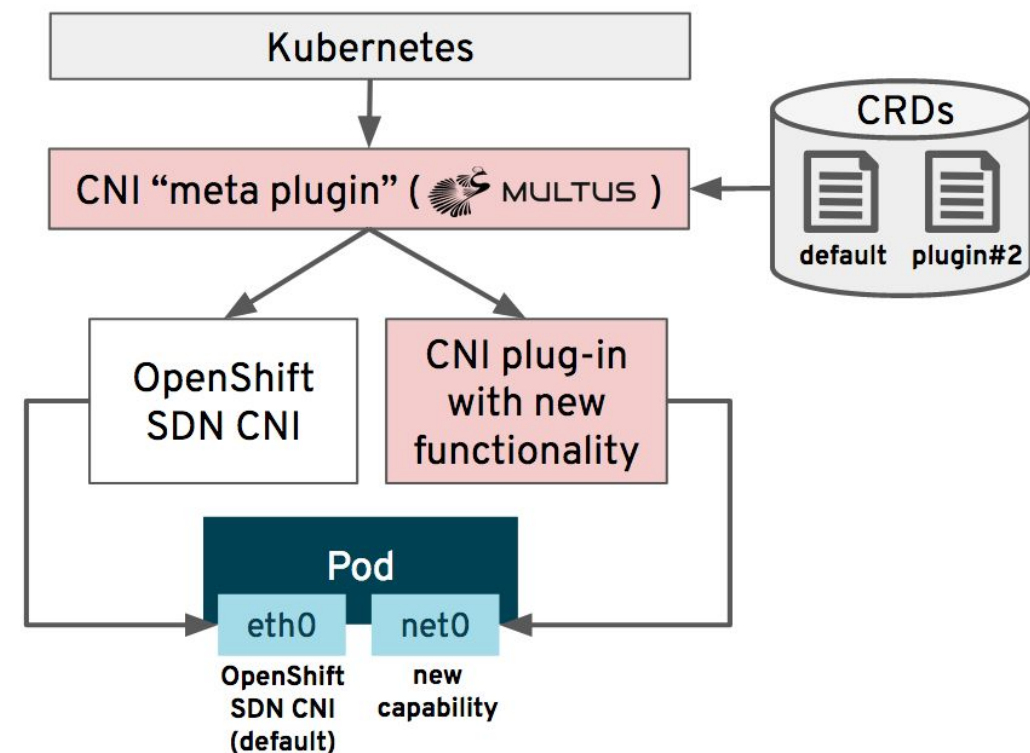
- Inherit many features and functions from Kubernetes
  - Scheduling, high availability, attach/detach resources
- Containerized virtual machines have the same characteristics as non-containerized
  - Resource (CPU, mem) limitations dictated by RHEL KVM
  - Linux and Windows guest operating systems
- Storage
  - Use Persistent Volumes Claims (PVCs) for VM disks
  - Clone and snapshots offloaded to CSI drivers
- Network
  - Inherit pod network by default
  - Multus enables direct connection to external network



# Network

# Virtual Machine Networking

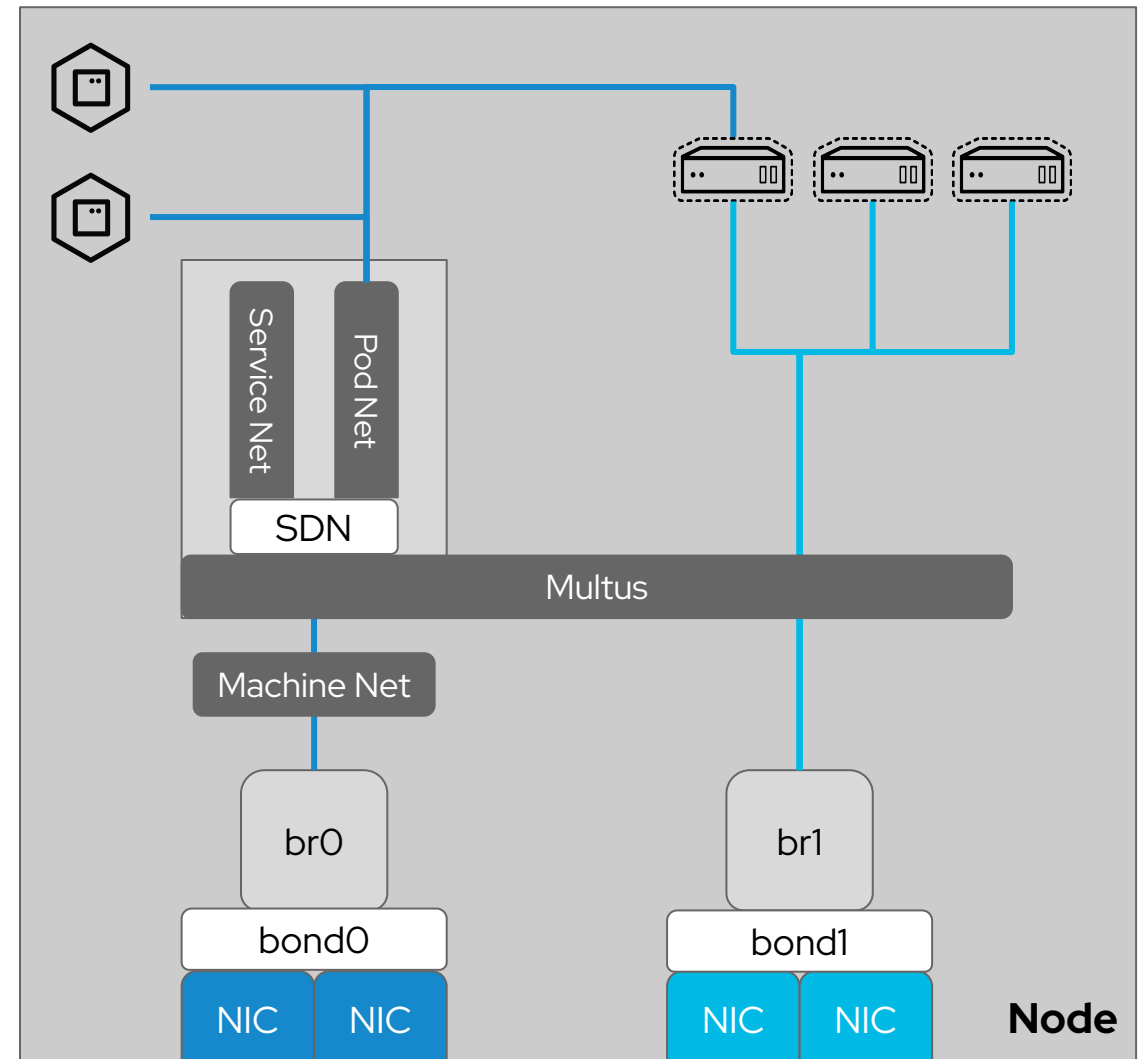
- Virtual machines optionally connect to the standard pod network
  - OpenShift SDN, OVNKubernetes
  - Partners, such as Calico, are also supported
- Additional network interfaces accessible via Multus:
  - Bridge, SR-IOV, OVN secondary networks
  - VLAN and other networks can be created at the host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





- Pod, service, and machine network are configured by OpenShift automatically
  - Use kernel parameters (dracut) for configuration at install - **bond0** in the example to the right
- Use the NMstate Operator to configure additional host network interfaces
  - **bond1** and **br1** in the example to the right
- VMs and Pods connect to one or more networks simultaneously

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



# GUI-based host network configuration

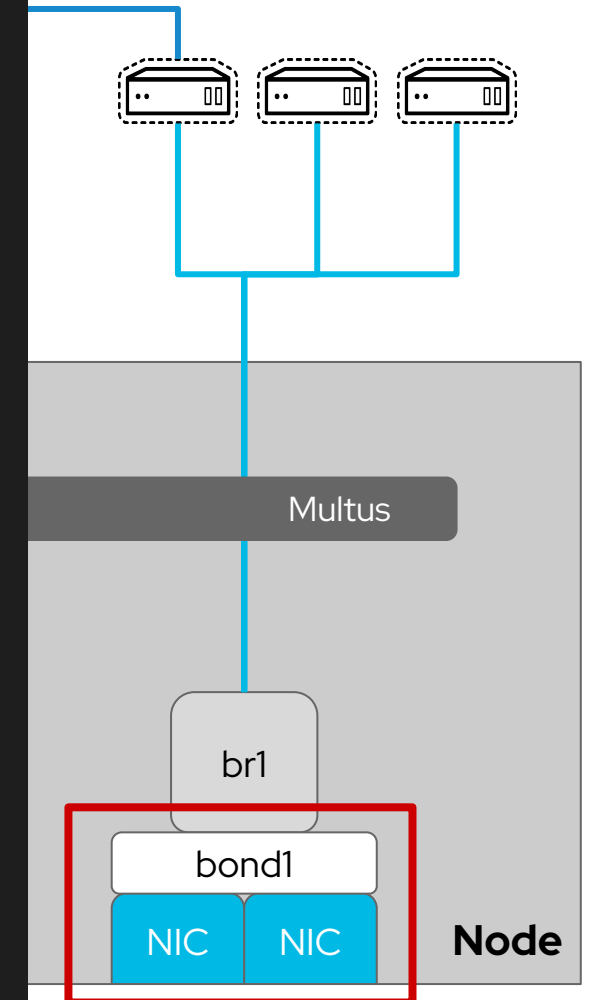
- Apply NMstate configuration using a form in the OpenShift admin console
- Create and configure
  - Ethernet interface IP (static, DHCP)
  - Bonds - mode 1-6 bonds with options, including IP configuration
  - Linux bridge configuration utilizing ethernet and/or bonds for “uplinks”
- Specify node selectors to have configuration automatically applied to matching nodes

The screenshot displays the Red Hat OpenShift Admin Console interface. On the left is a sidebar menu with categories like Administrator, Home, Operators, Workloads, Virtualization, Migration, Networking, Services, Routes, Ingresses, NetworkPolicies, NetworkAttachmentDefinitions, NodeNetworkConfigurationPolicy (highlighted), NodeNetworkState, Storage, Builds, Observe, Compute, User Management, and Administration. The main panel is titled 'Create NodeNetworkConfigurationPolicy' with an 'Edit YAML' link. It contains a description of NMstate configuration, a checkbox to apply the policy to specific node subsets (checked), a 'Policy name' field with 'worker-bond0-br1', a 'Description' field, and a 'Policy Interface(s)' section. This section lists 'Bonding bond0' and 'Bridge br1'. The 'Bridge br1' details include 'Interface name' (br1), 'Network state' (Up), 'Type' (Bridge), 'IP configuration' (IPv4), 'Port' (bond0), and an 'Enable STP' checkbox. A 'Create' button is at the bottom.

# Host bond configuration

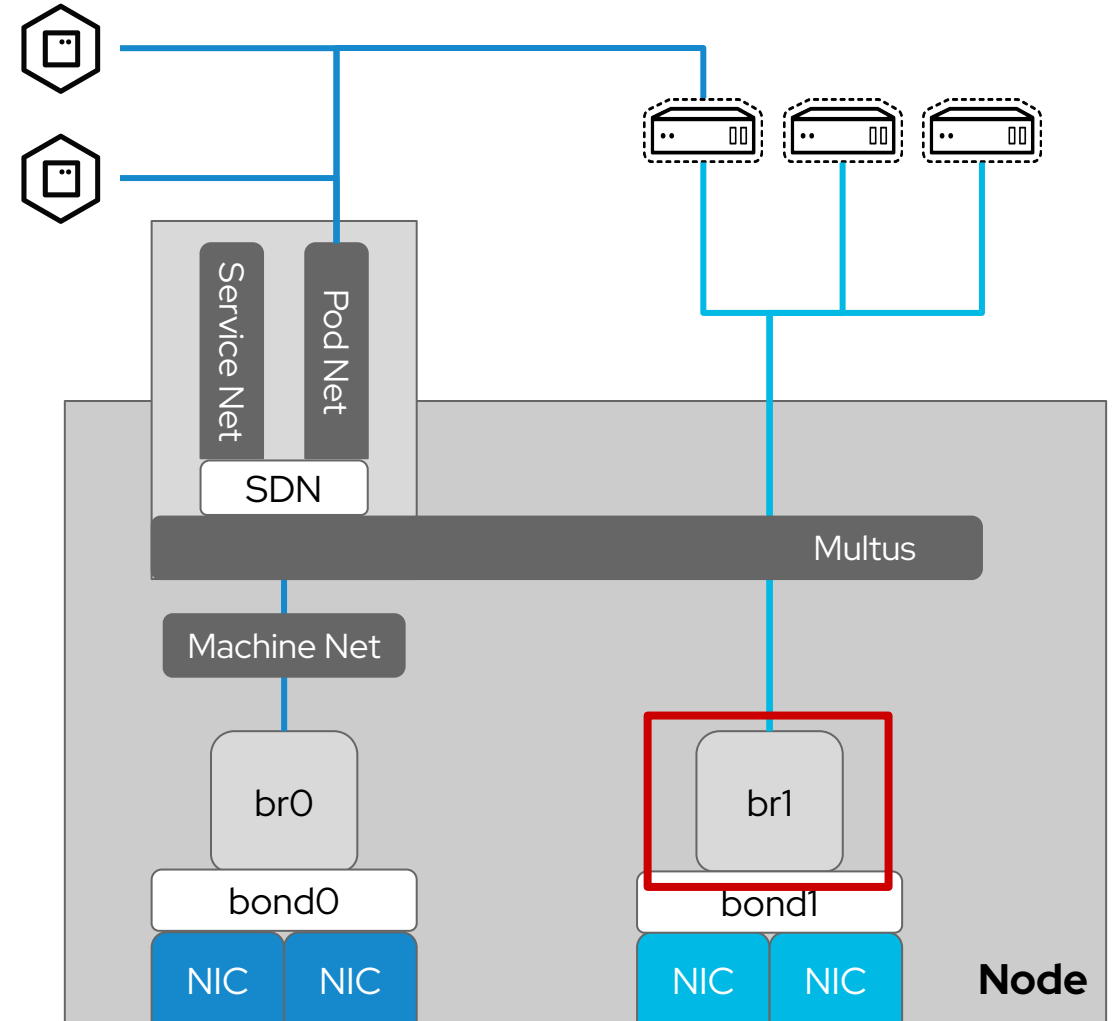
- NodeNetworkConfigurationPolicy (NNCP)
  - Nmstate operator CRD
  - Configure host network using declarative language
- Applies to all nodes specified in the nodeSelector, including newly added nodes automatically
- Update or add new NNCPs for additional host configs

```
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

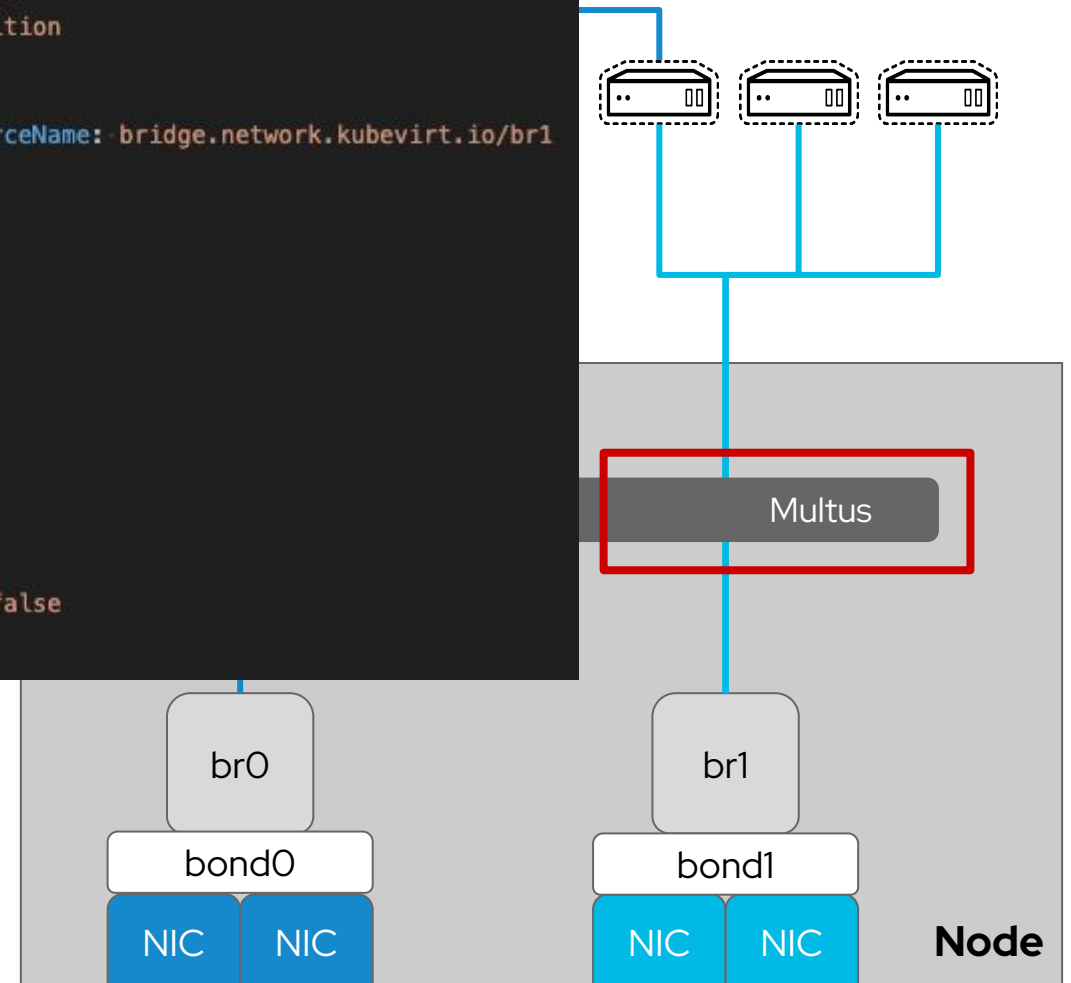
```
1  apiVersion: nmstate.io/v1alpha1
2  kind: NodeNetworkConfigurationPolicy
3  metadata:
4    name: worker-bond1-br1
5  spec:
6    nodeSelector:
7      node-role.kubernetes.io/worker: ""
8    desiredState:
9      interfaces:
10       - name: br1
11         description: br1 with bond1
12         type: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



# Network Attachment Definition configuration

- net-attach-def configures multus to allow the VM to access an underlying resource
  - Optionally define VLAN tags
- Limited to the namespace it's created in
  - Except the default namespace, which is available to all

```
1  apiVersion: k8s.cni.cncf.io/v1
2  kind: NetworkAttachmentDefinition
3  metadata:
4    annotations:
5      k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
6    name: vlan-93
7    namespace: default
8  spec:
9    config: >-
10    - {
11      "name": "vlan-93"
12      "type": "cnv-bridge",
13      "cniVersion": "0.3.1",
14      "bridge": "br1",
15      "vlan": 93,
16      "macspoofchk": true,
17      "ipam": {},
18      "preserveDefaultVlan": false
19    }
```



# Host network configuration status

- Use the admin console to view the NodeNetworkState (OpenShift 4.14+)
- Detailed configuration information for host networking including
  - IP and MAC addresses
  - Bond configuration
  - Bridge configuration
- Review and troubleshoot host network configuration

The screenshot displays the Red Hat OpenShift Admin Console interface. On the left sidebar, the 'NodeNetworkState' option is selected, indicated by a red circle with the number 1. The main content area shows the 'NodeNetworkState' page for a specific node, indicated by a red circle with the number 2. The page title is 'NodeNetworkState'. Below the title, there is a search bar and a table of network interfaces. The table has columns for 'Name', 'IP address', 'Ports', and 'MAC address'. The table lists two network interfaces: 'ethernet' and 'linux-bridge'. The 'ethernet' interface has three sub-interfaces: 'enp1s0', 'enp2s0', and 'enp3s0'. The 'linux-bridge' interface has one sub-interface: 'br-flat'. A red circle with the number 3 points to the 'linux-bridge' section of the table.

Name	IP address	Ports	MAC address
ethernet			
enp1s0	172.22.0.71/24	-	DE:AD:BE:EF:00:04
enp2s0	192.168.123.104/24	-	52:54:00:00:00:04
enp3s0	-	-	52:54:00:00:01:04
linux-bridge			
br-flat	-	1	52:54:00:00:01:04

# 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 easier*** 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

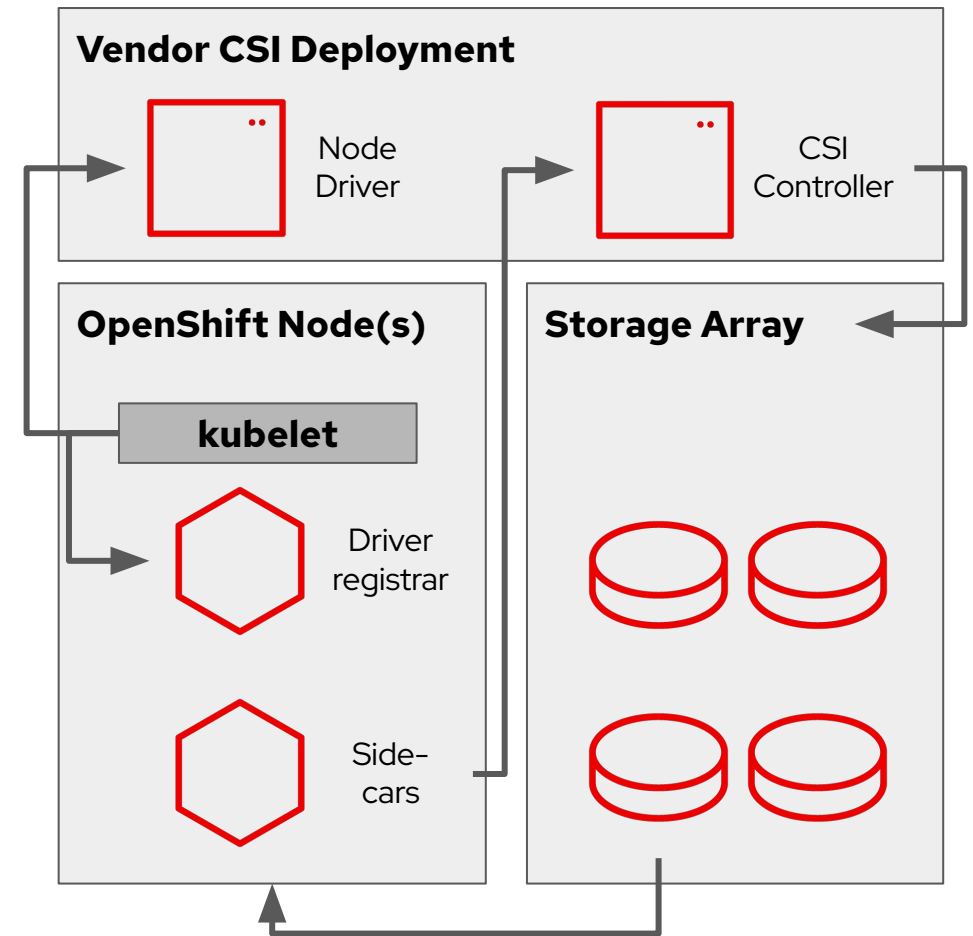
# Virtual Machine Storage

- OpenShift Virtualization uses the Kubernetes PersistentVolume (PV) paradigm
- PVs can be backed by
  - CSI drivers, including partners and ODF
  - Local storage using host path provisioner / logical volume Operator
- Use dynamically or statically provisioned PVs
- RWX is **required** for live migration
- Disks are attached using VirtIO or SCSI controllers
  - Connection order specified in the VM definition
- Boot order customized via VM definition

PersistentVolumeClaim Details	
<b>Name</b> rhel-rootdisk	<b>Status</b> ✔ Bound
<b>Namespace</b> NS default	<b>Capacity</b> 20Gi
<b>Labels</b> app=containerized-data-importer	<b>Access Modes</b> ReadWriteMany
<b>Annotations</b> 12 Annotations	<b>Volume Mode</b> Filesystem
<b>Label Selector</b> No selector	<b>Storage Class</b> SC managed-nfs-storage
<b>Created At</b> Jul 8, 4:18 pm	<b>Persistent Volume</b> PV pvc-a1aac411-2e46-495a-897e-cf3bc2442199
<b>Owner</b> DV rhel-rootdisk	

# VM disks in PVCs

- VM disks on FileSystem mode PVCs are created as thin provisioned raw images by default, but can be configured
- Block mode PVCs are attached directly to the VM
  - Many partners support RWX with block mode PVCs
- CSI features, e.g. snapshot and clone, offload operations to the storage device
  - Use DataVolumes to clone VM disks to automatically select the optimal method to clone the disk
  - Use VM details interface for (powered off) VM snaps
- Resize VM disks using standard PVC methods
- Hot add is not supported



# 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
- VM disks are cloned / copied from existing PVCs
- DataVolumes are created as distinct objects or as a part of the VM definition as a `dataVolumeTemplate`
- DataVolumes use the `ContainerizedDataImporter` to connect, download, and prepare the disk image
- DataVolumes create PVCs based on defaults defined in the profile
  - Access mode, snapshot method

```
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'
```

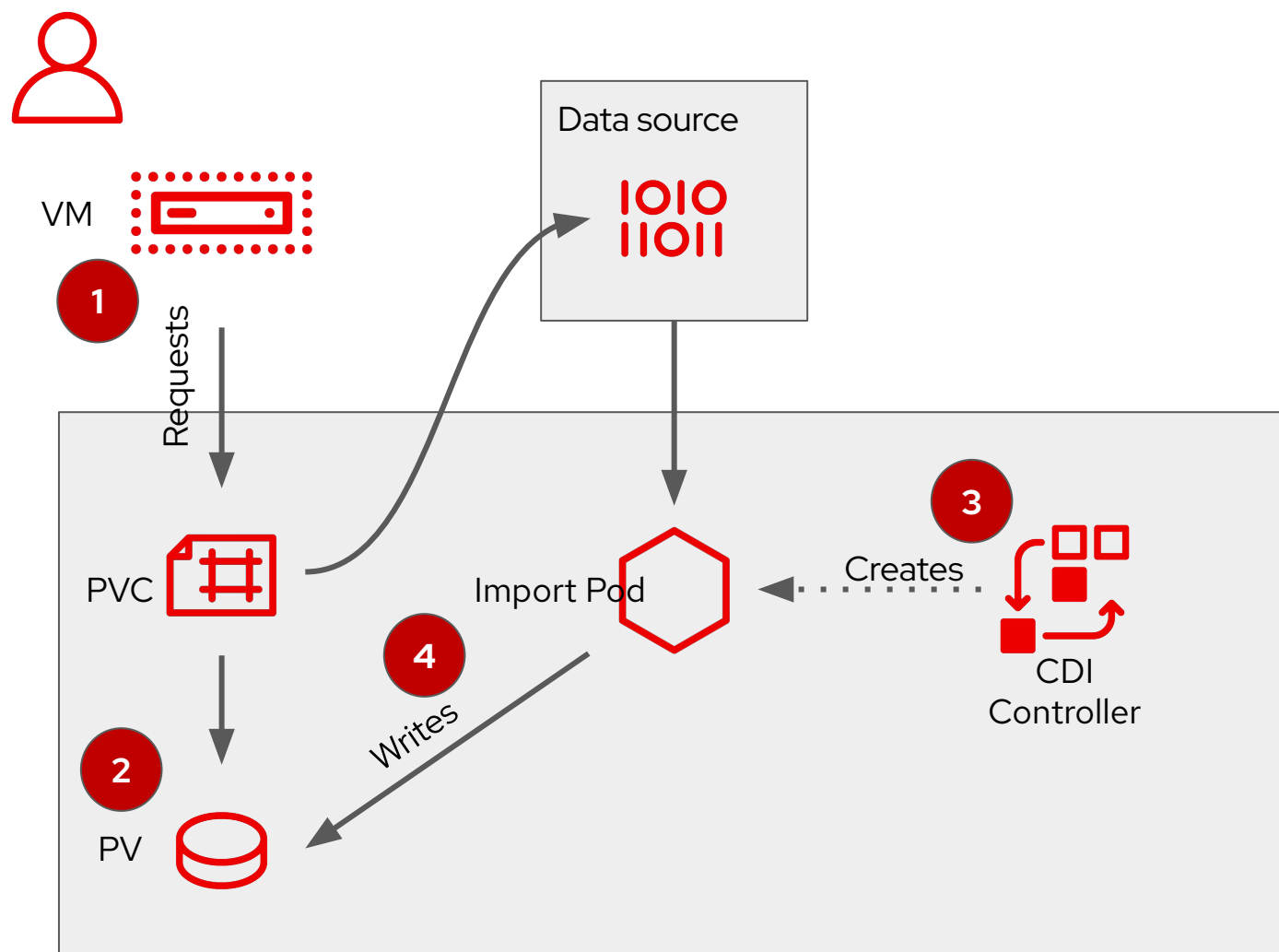
# Storage Profiles

- Provide default settings and properties for StorageClasses used by DataVolumes
- Created automatically for every StorageClass
- Preconfigured values for many storage providers, administrator can modify and customize
- DataVolume definitions only need to specify StorageClass, without knowledge of underlying details
  - spec.storage doesn't require fields other than the size and StorageClass

```
1  apiVersion: cdi.kubevirt.io/v1beta1
2  kind: StorageProfile
3  metadata:
4    name: hostpath-provisioner
5  spec:
6    claimPropertySets:
7      - accessModes:
8          - ReadWriteOnce
9        volumeMode:
10         Filesystem
```

```
1  apiVersion: cdi.kubevirt.io/v1alpha1
2  kind: DataVolume
3  metadata:
4    name: rhel8-vm-disk
5  spec:
6    storage:
7      resources:
8        requests:
9          storage: 30Gi
10     storageClassName: hostpath-provisioner
11    source:
12      pvc:
13        name: rhel8-cloud-image
14        namespace: os-images
```

# 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 node 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.1an: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**
- 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
- Uses the SDN by default, customizable to a dedicated network after install

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

# Automated live migration

- OpenShift / Kubernetes triggers Pod rebalance actions based on multiple factors
  - Soft / hard eviction policies
  - Pod descheduler
  - Pod disruption policy
  - Node resource contention resulting in evictions
    - Pods are `Burstable` QoS class by default
    - All memory is requested in Pod definition, only CPU overhead is requested
- Pod rebalance applies to VM pods equally
- VMs will behave according to the eviction strategy
  - `LiveMigrate` - use live migration to move the VM to a different node
  - No definition - terminate the VM if the node is drained or Pod evicted

# 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
  - Large / Huge page support for VM memory
  - Use scheduler profiles to provide additional hints (for all Pods)
- 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) allows 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
- Pods request full amount of VM memory and approx. 10% of VM CPU
  - VM pods request a small amount of additional memory, used for libvirt/QEMU overhead
    - Administrator can set this to be overcommitted

# High availability

- Node failure is detected by Kubernetes and results in the Pods from the lost node being rescheduled to the surviving nodes
  - Use machine health checks, node health checks, and the Node Self Remediation Operator to expedite detection and recovery
- 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
  - Machine health checks can decrease failure detection time

# Terminology comparison

Feature	RHV	OpenShift Virtualization	vSphere
<b>Where VM disks are stored</b>	Storage Domain	PVC	datastore
<b>Policy based storage</b>	None	StorageClass	SPBM
<b>Non-disruptive VM migration</b>	Live migration	Live migration	vMotion
<b>Non-disruptive VM storage migration</b>	Storage live migration	N/A	Storage vMotion
<b>Active resource balancing</b>	Cluster scheduling policy	Pod eviction policy, descheduler	Dynamic Resource Scheduling (DRS)
<b>Physical network configuration</b>	Host network config (via nmstate w/4.4)	nmstate Operator, Multus	vSwitch / DvSwitch
<b>Overlay network configuration</b>	OVN	OCP SDN (OpenShiftSDN, OVNKubernetes, and partners), Multus	NSX-T
<b>Host / VM metrics</b>	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 project template so that non-VM pods are not overcommitted
  - Customize projects hosting VMs for overcommit policy
- Apply Quota and LimitRange controls to projects with VMs to manage resource consumption
- VM definitions default to all memory “reserved” via a request, but only a small amount of CPU
  - CPU and memory request/limit values are modified in the VM definition

# 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.cidr` 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

- Shared storage is not required, but *very highly encouraged*
  - **Live migration depends on RWX PVCs**
- Create shared storage from local resources using 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
  - Many non-certified CSI provisioners work, but do not have same level of OpenShift testing
- Local storage may be utilized via the Host Path Provisioner

# 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
    - Install to a PVC using an ISO
  - 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 Virtualization content kit: <https://red.ht/virtkit>
- Documentation:
  - OpenShift Virtualization: <https://docs.openshift.com>
  - KubeVirt: <https://kubevirt.io>
- Demos and video resources:  
[https://www.youtube.com/playlist?list=PLaR6Rq6Z4lqeQeTosfoFzTyE\\_QmWZW6n](https://www.youtube.com/playlist?list=PLaR6Rq6Z4lqeQeTosfoFzTyE_QmWZW6n)
- Labs and workshops: <https://demo.redhat.com>



# The LAB Starts at 13:30 !!

<https://demo.redhat.com/workshop/pmfaa6>

Password:

D54ha4s2

# 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://linkedin.com/company/red-hat)

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

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

 [twitter.com/RedHat](https://twitter.com/RedHat)