Red Hat
OpenShift

OpenShift Virtualization

Yeni Nesil Sanallaştırma

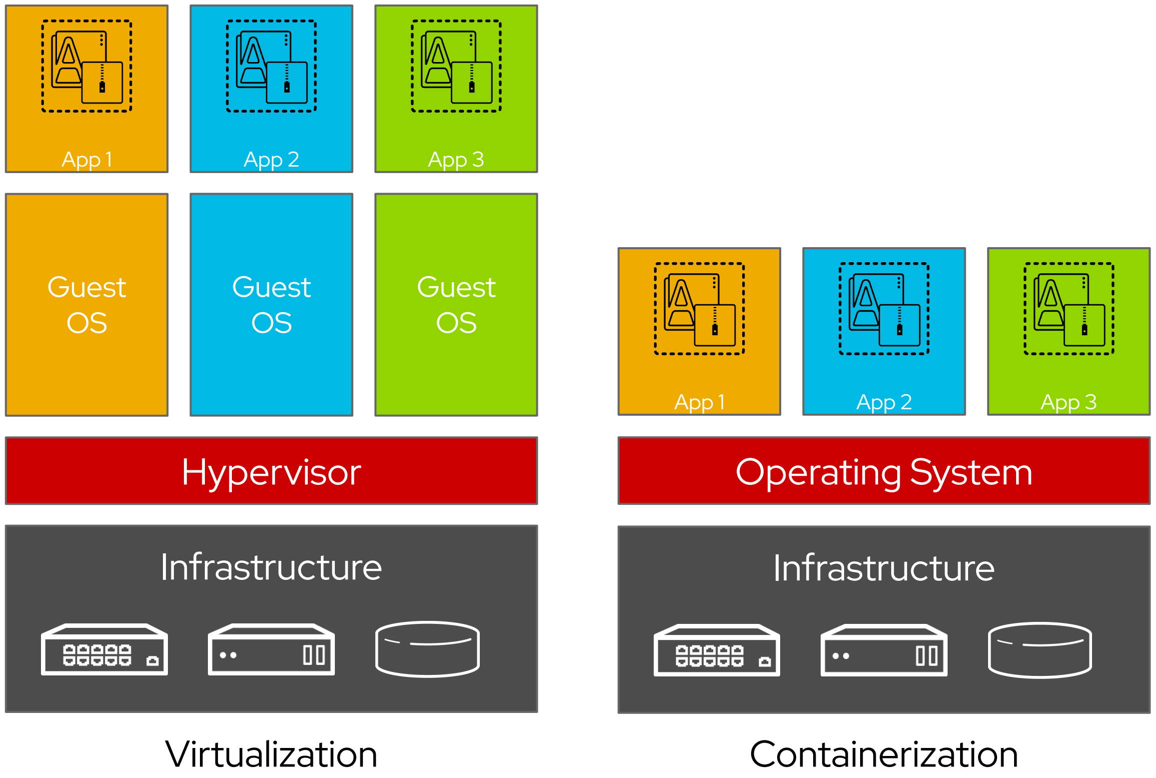
Koray Şeremet
Eylül, 2020

Red Hat

What is OpenShift Virtualization?

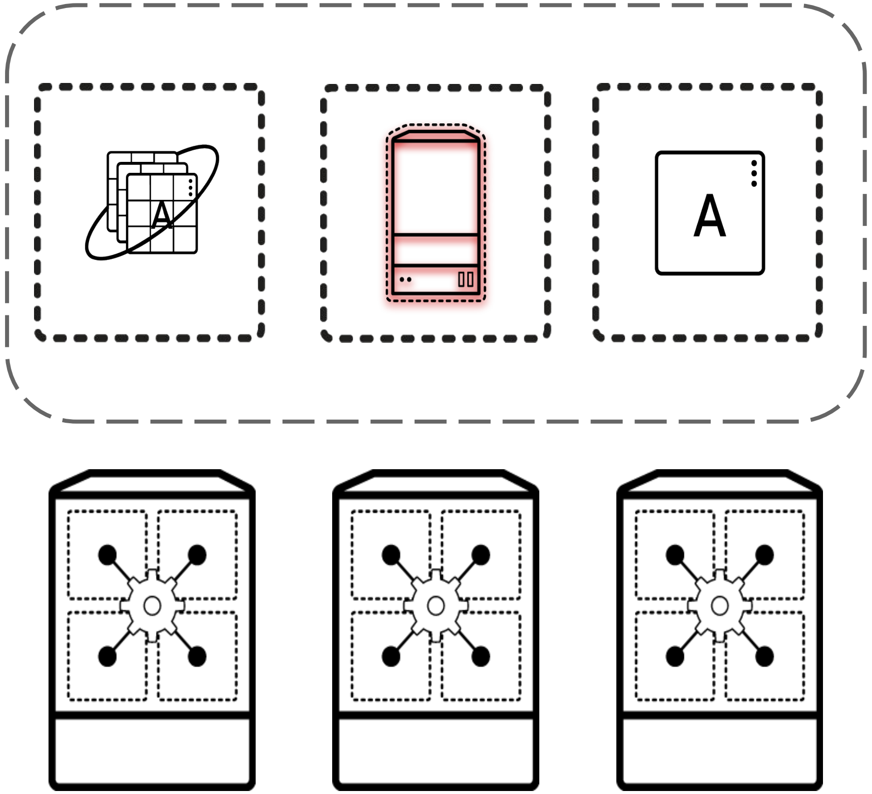
Containers are not virtual machines

- 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



V0000000

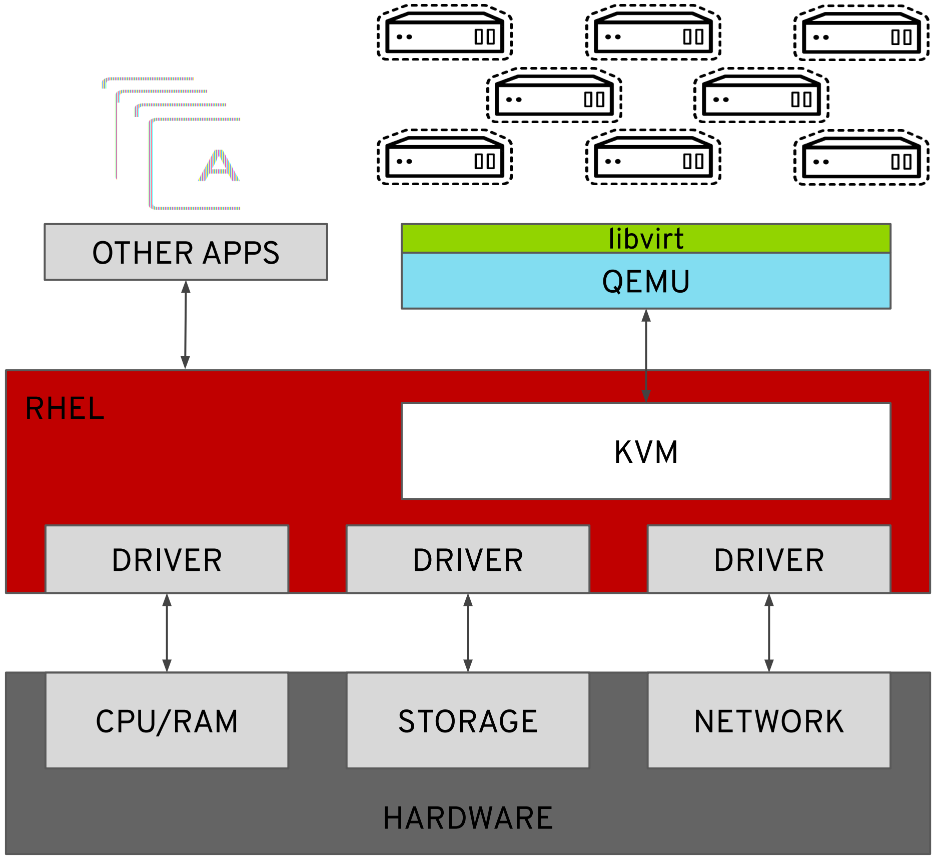
Red Hat

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

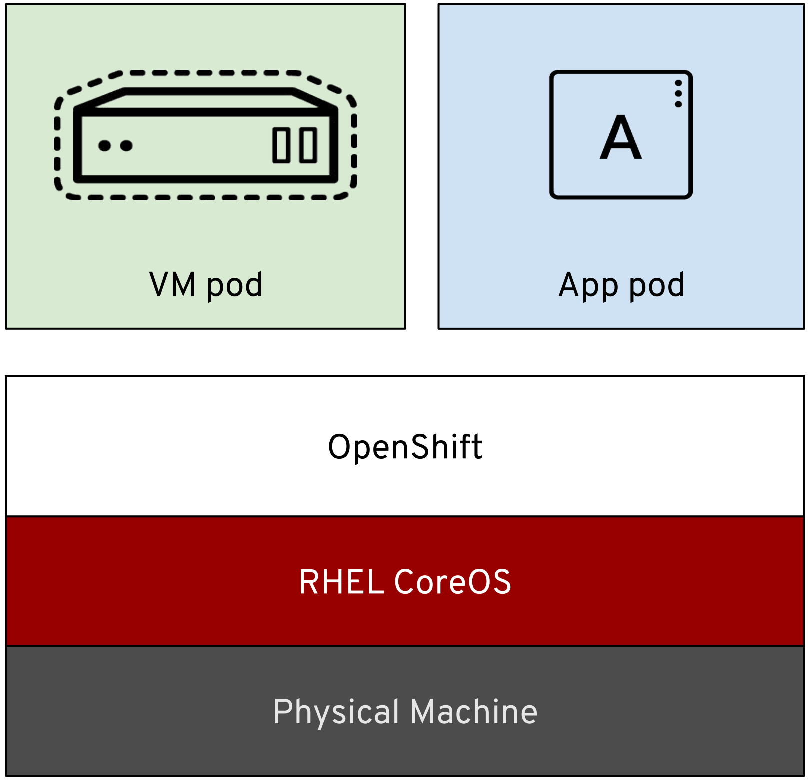
- OpenShift Virtualization uses KVM, the Linux kernel hypervisor
- KVM is a core component of the Red Hat Enterprise Linux kernel
  - KVM has 10+ years of production use: Red Hat Virtualization, Red Hat OpenStack Platform, and RHEL all leverage KVM, QEMU, and libvirt
- QEMU uses KVM to execute virtual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

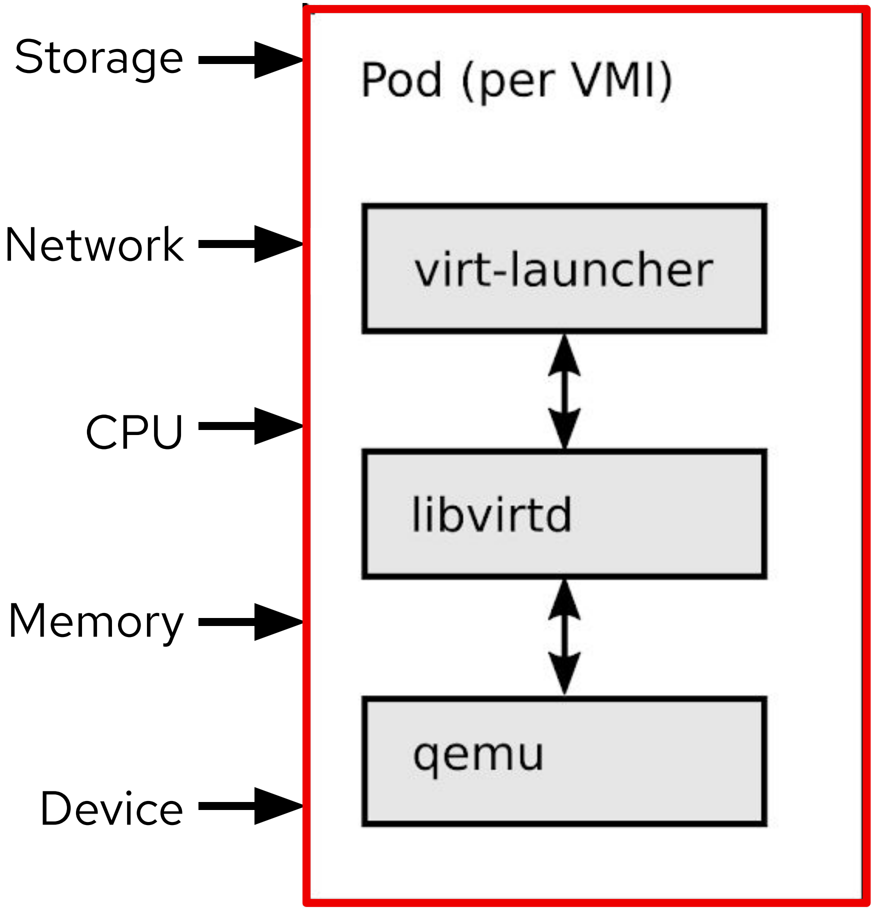


Virtualization native to Kubernetes

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

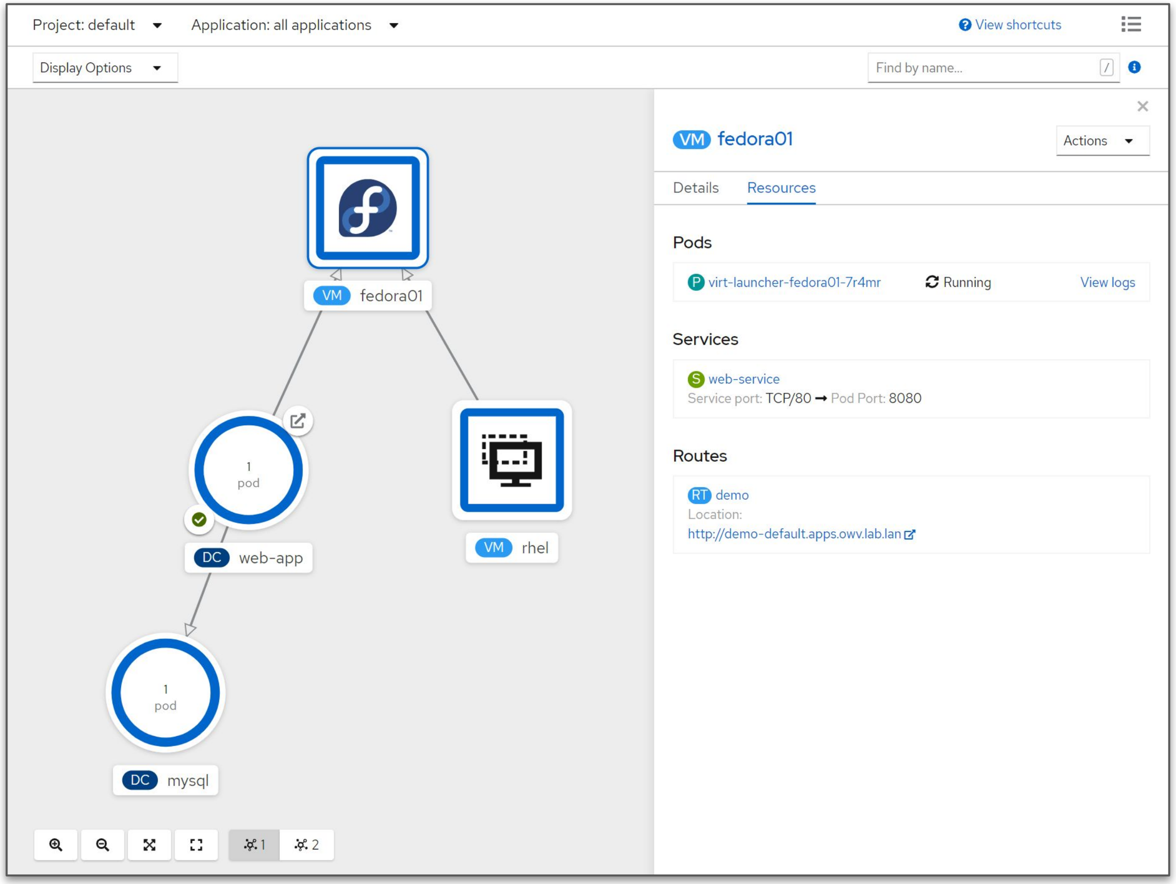
- libvirt and qemu from RHEL are mature, have high performance, provide stable abstractions, and have a minimal overhead.

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



Demo

Create VMs

Import VMs

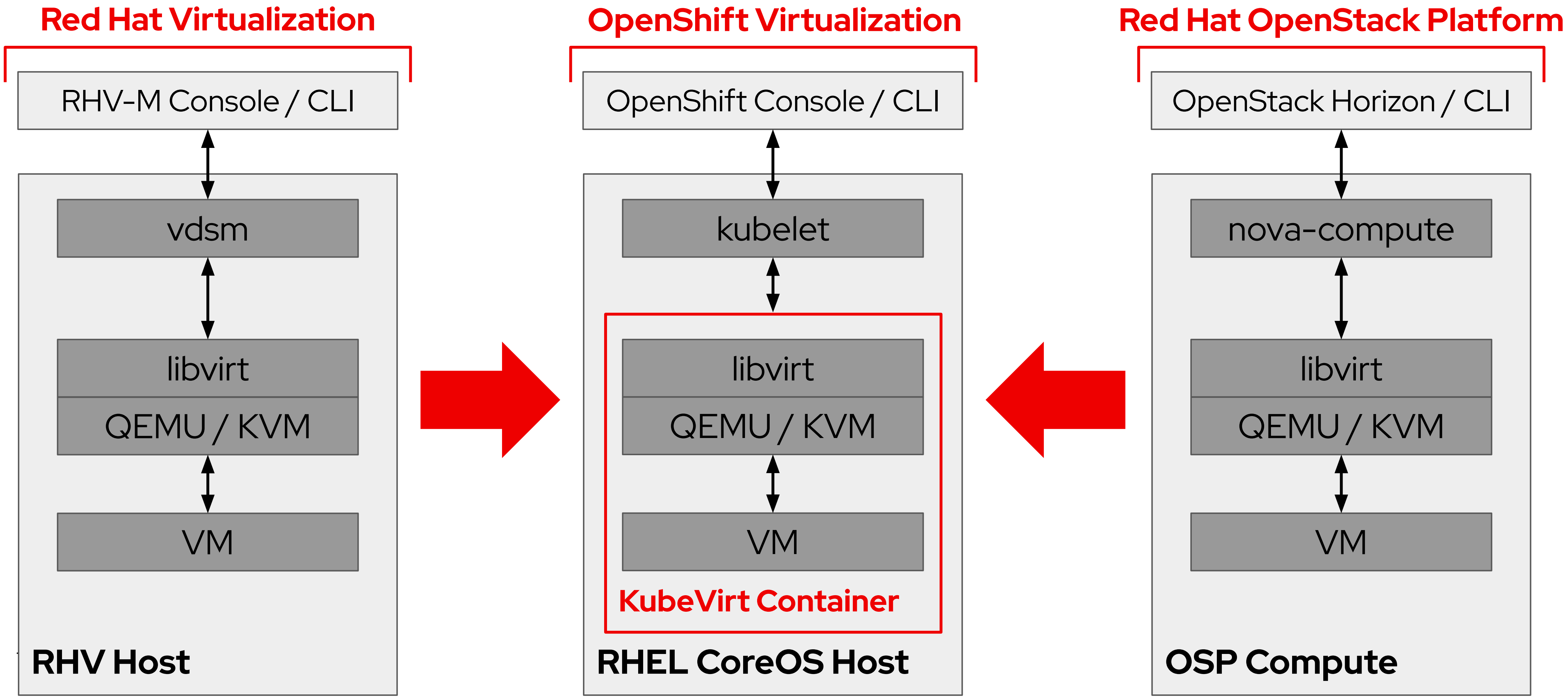
View / manage VMs

Destroy VMs

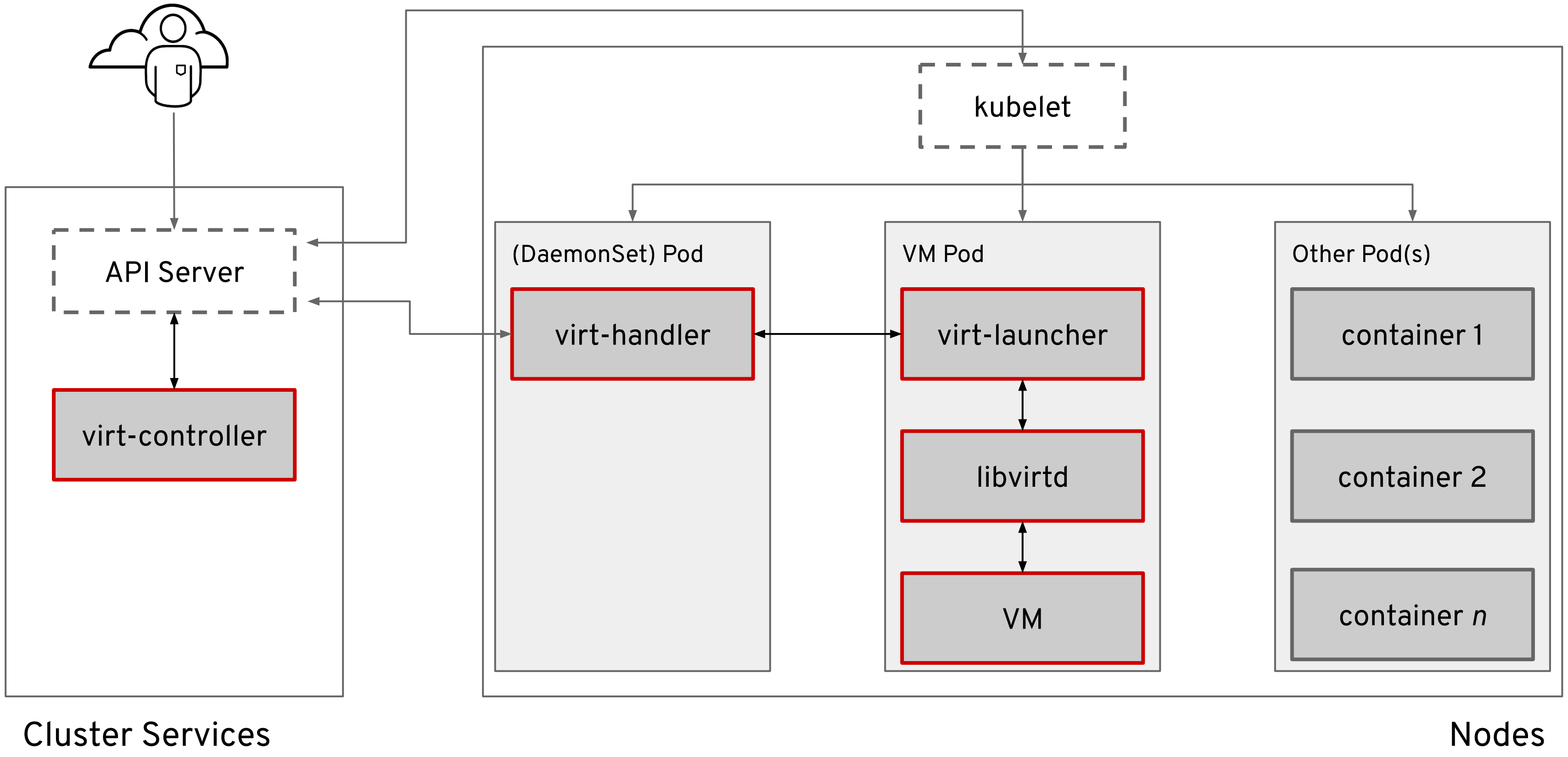
Metrics

Deeper into the technology

Containerizing KVM



Architectural Overview



Reference: https://kubernetes.io/blog/2018/05/22/getting-to-know-kubevirt/

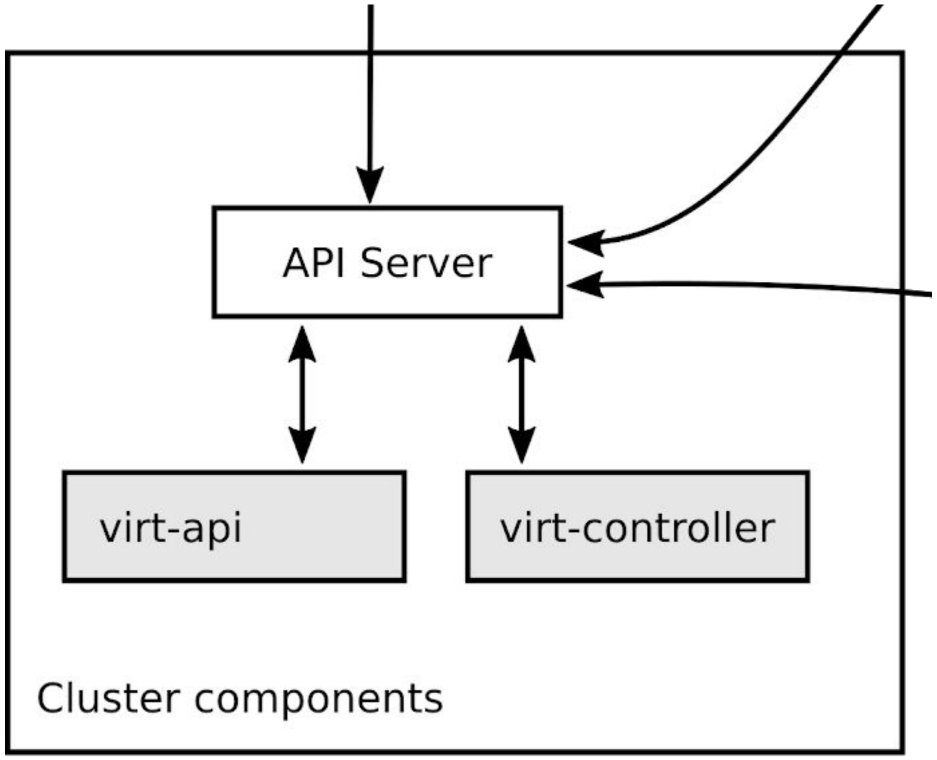
Adding virtualization to the Kubernetes API

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

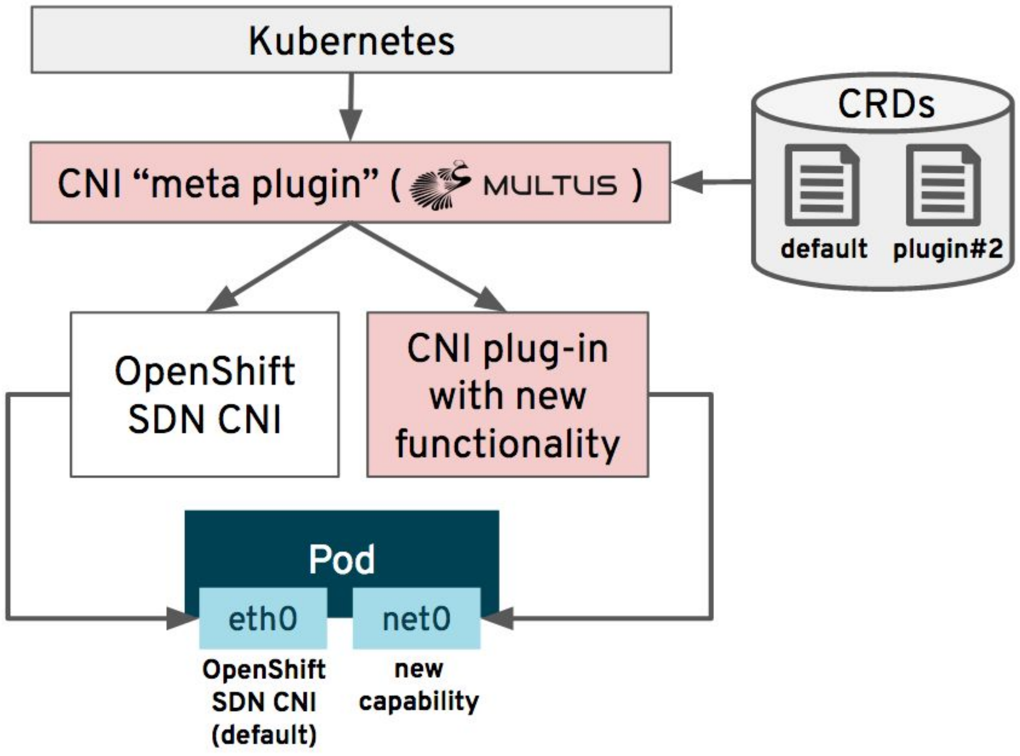


Red Hat

Network

Virtual Machine Networking

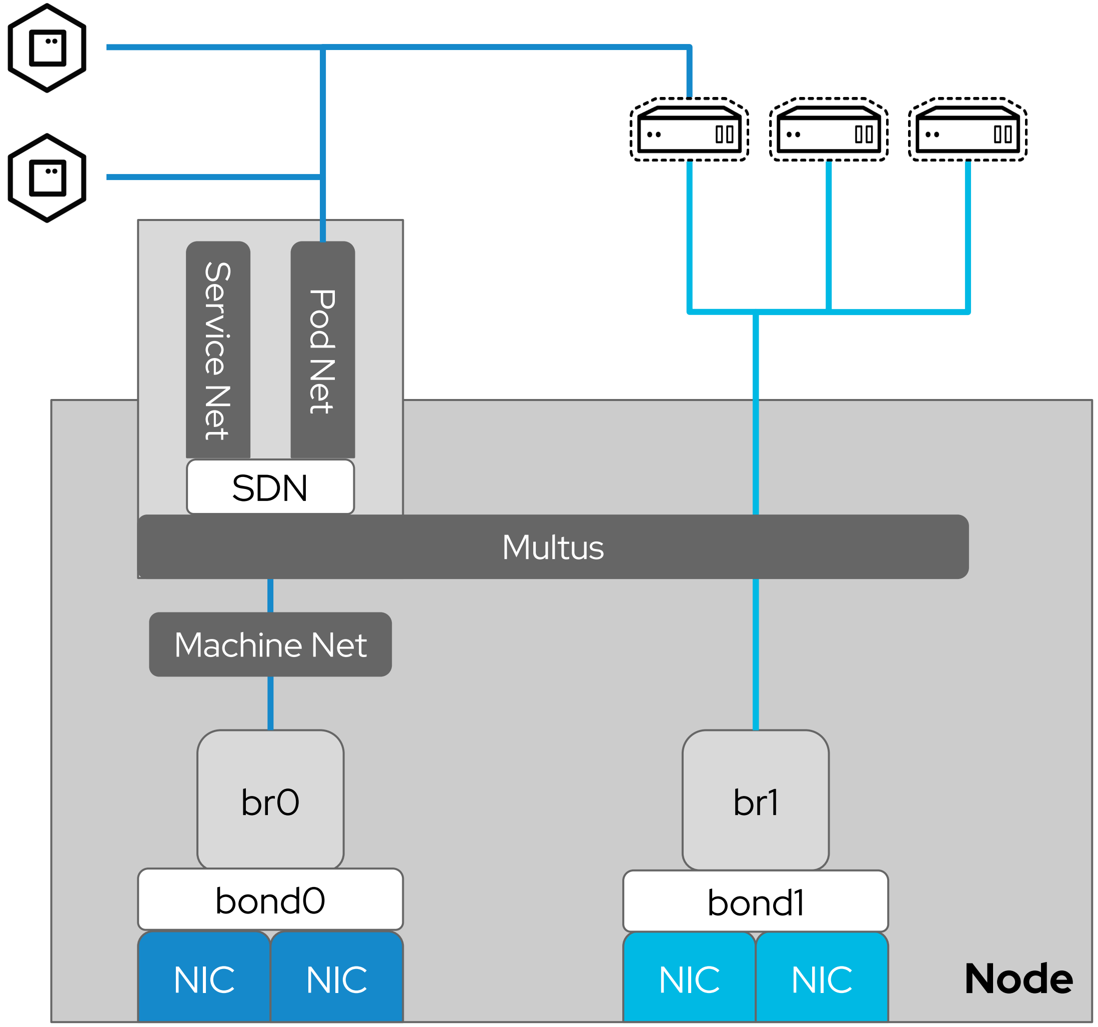
- Virtual machines optionally connect to the standard pod network
  - OpenShift SDN, OVNKubernetes, etc.
- Additional network interfaces accessible via Multus:
  - Bridge, SR-IOV
  - VLAN and other networks can be created using nmstate at the host level
- When using at least one interface on the default 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 `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

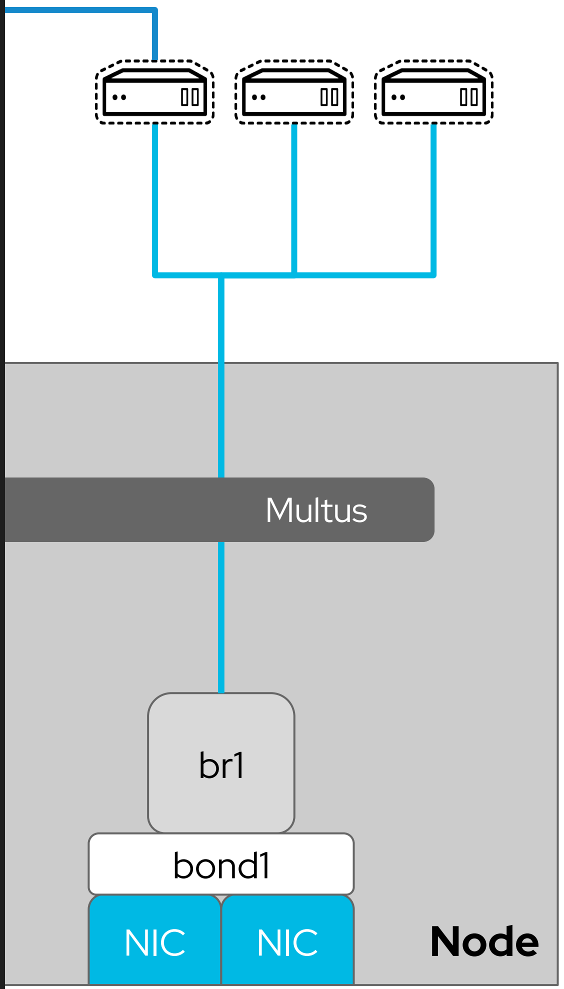
The following slides show an example of how this setup is configured



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

```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
```



Connecting VMs to networks

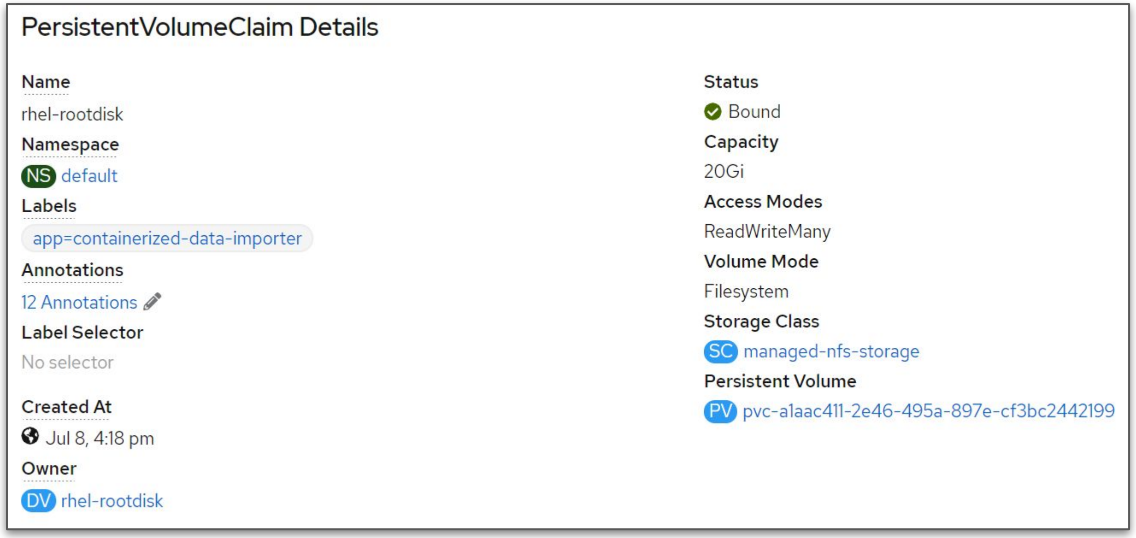
- Virtual machine interfaces describe NICs attached to the VM
  - `spec.domain.devices.interfaces`
  - Model: virtio, e1000, pcnet, rtl8139, etc.
  - Type: masquerade, bridge
  - MAC address: customize the MAC
- The networks definition describes the connection type
  - `spec.networks`
  - Pod = default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

```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
  - 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

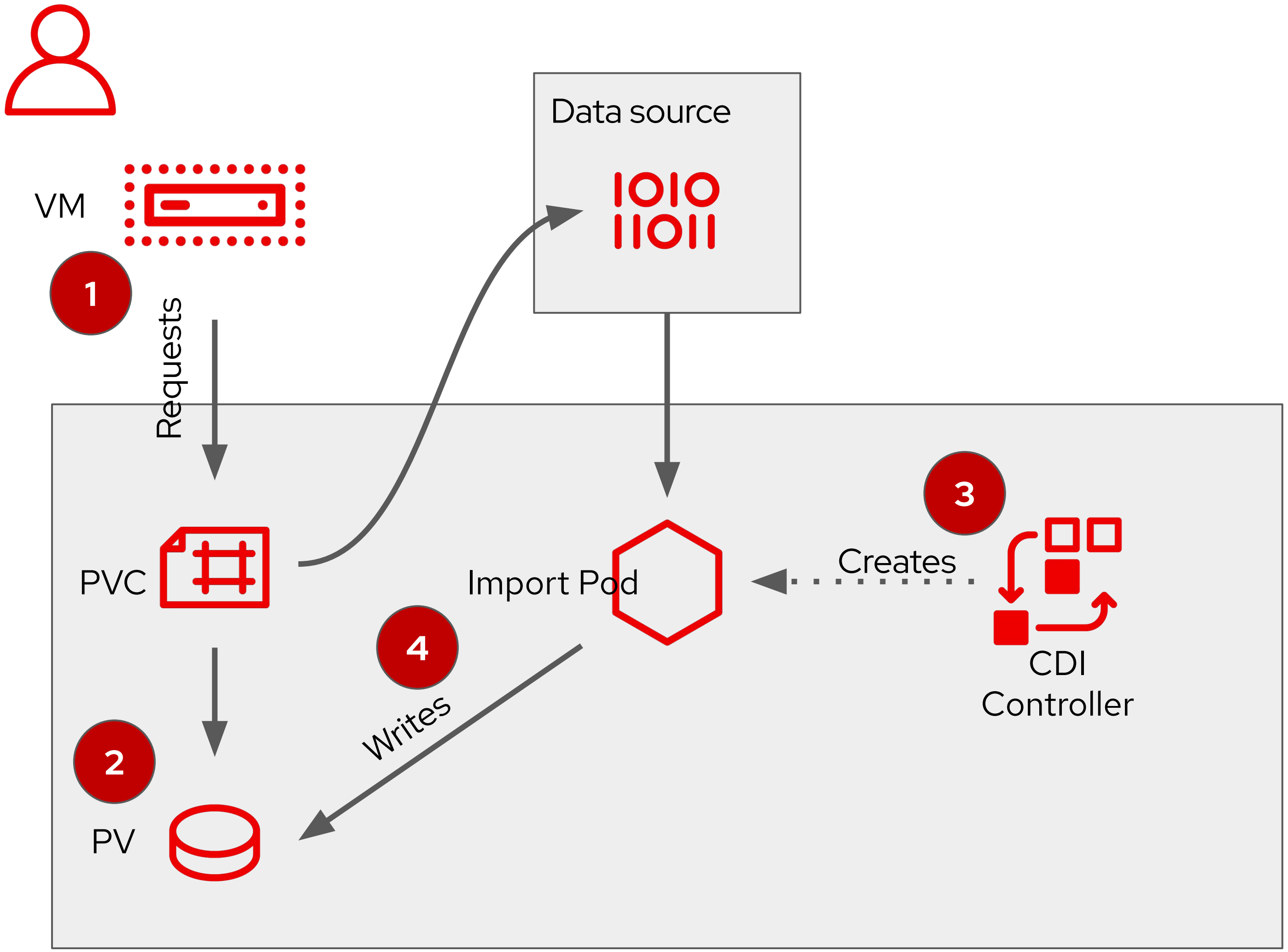


DataVolumes

- 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	-

Demo

Create Linux VM Replica Sets

Using ConfigMaps

Monitoring VM health

Create Windows VM Replica Sets

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

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 match VM requirements - `Guaranteed` QoS by default
  - K8s QoS policy determines scheduling priority: `BestEffort` class is evicted before `Burstable` class, which is evicted before `Guaranteed` class

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

Reference: https://kubevirt.io/user-guide/#/installation/unresponsive-nodes

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

Additional resources

More information

- Documentation:
  - OpenShift Virtualization: https://docs.openshift.com
  - KubeVirt: https://kubevirt.io
- Demos and video resources: http://demo.openshift.com
- Recent blog article: https://red.ht/3b9cdwH
- Demo resources used in this webinar: https://github.com/kseremet/kubevirt-webinar.git

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