OpenShift Virtualization

Technical Deep Dive

Alfred Bach
PSA Red Hat
Field Partner and Learning Team

09:00 Welcome and Intro

09:15 OpenShift Basic installation on Bare Metal Cluster Layout discussion Compact, Single Node, Regular, HCP ….

10:30 Installing the OpenShift Virtualisation Operator and configuration

11:00 BREAK

11:15 Setting up Local storage (Single Node and OpenShift Data Foundation) Networking options, installing NMStat and SR-IOV

12:00 Lunch Break

13:00 Backup and Disaster Recovery (OADP and ODF)

13:30 Migration Options (MTV and Ansible Migration)

14:30 Migration Risks

15:00 Q&A

Install OpenShift Container Platform (OCP)

OpenShift 4.16 Supported Providers

Installation Experiences

aws Outposts
AWS Local Zones
Azure
IBM Cloud
Azure Stack Hub
IBM Power Systems
Alibaba Cloud
IBM Z
Bare Metal
NUTANIX
Google Cloud
RED HAT OPENSTACK PLATFORM
vmware vSphere
RED HAT VIRTUALIZATION

Full Stack Automation

Installer Provisioned Infrastructure

- Auto-provisions infrastructure
- *KS like
- Enables self-service

Pre-existing Infrastructure

User Provisioned Infrastructure

- Bring your own hosts
- You choose infrastructure automation
- Full flexibility
- Integrate ISV solutions

Interactive – Connected

Assisted Installer

- Hosted web-based guided experience
- Agnostic, bare metal, and vSphere only
- ISO Driven

Local – Disconnected

Agent-based Installer

- Disconnected bare metal deployments
- Automated installations via CLI
- ISO driven

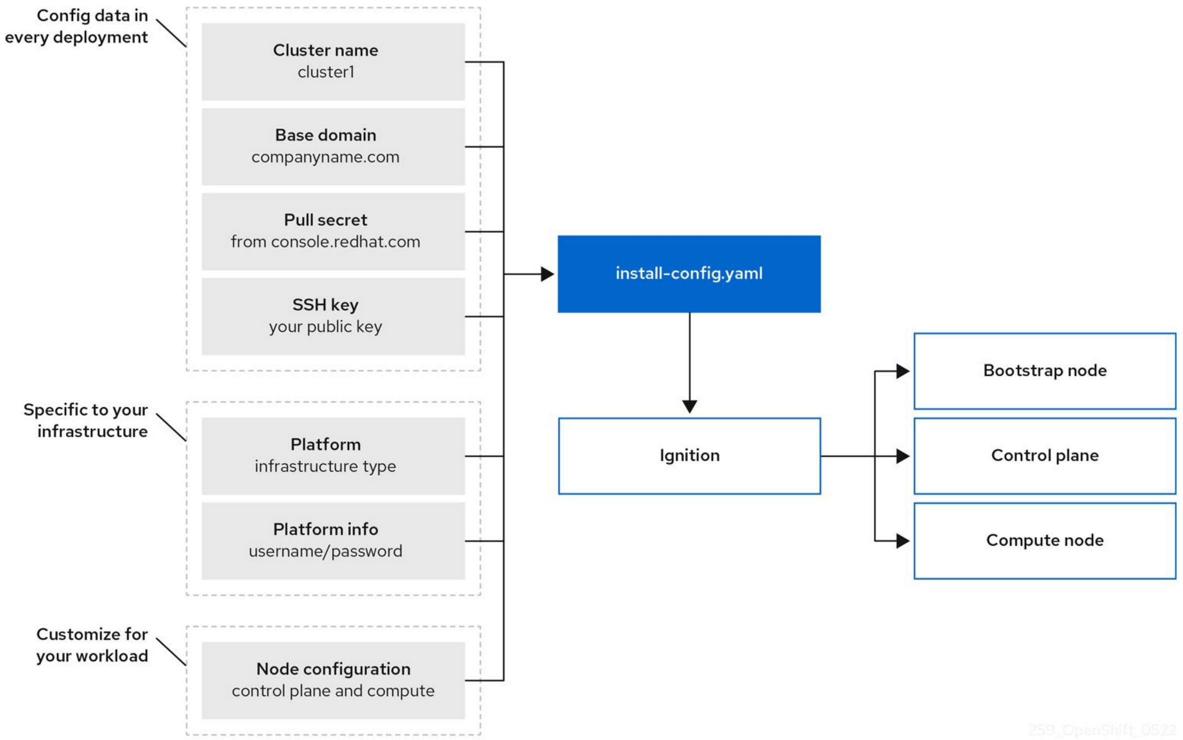
Red Hat

Product Manager(s): Marcos Entenza (AWS*, Azure*, GCP, IBM Cloud, Nutanix), Gaurav Singh (Alibaba), Ramon Acedo (BM, VMware), Peter Lauterbach (RHV & OCP Virtualization), Gil Rosenberg (OpenStack), & Duncan Hardie (IBM Z & Power)

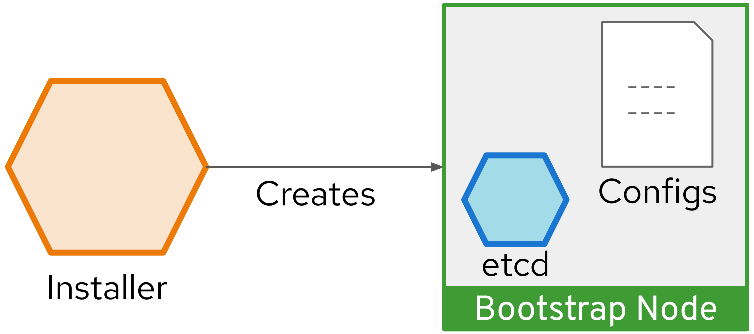
OpenShift Bootstrap Process: Self-Managed Kubernetes

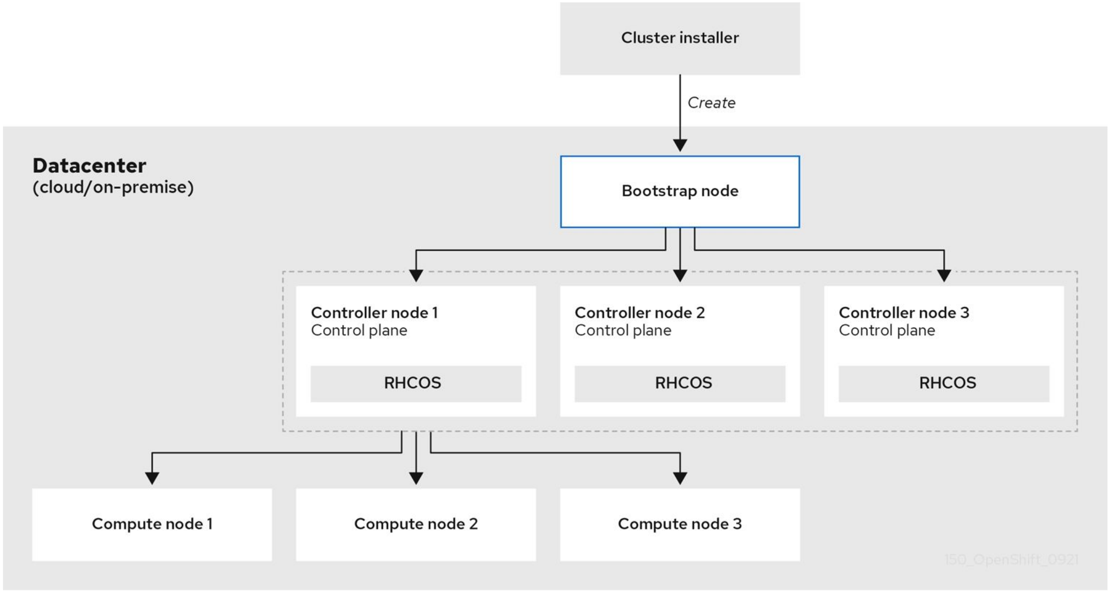
How to boot a self-managed cluster:

- OpenShift 4 is unique in that management extends all the way down to the operating system
- Every machine boots with a configuration that references resources hosted in the cluster it joins enabling cluster to manage itself
- Downside is that every machine looking to join the cluster is waiting on the cluster to be created
- Dependency loop is broken using a bootstrap machine, which acts as a temporary control plane whose sole purpose is bringing up the permanent control plane nodes
- Permanent control plane nodes get booted and join the cluster leveraging the control plane on the bootstrap machine
- Once the pivot to the permanent control plane takes place, the remaining worker nodes can be booted and join the cluster



OpenShift Bootstrap Process: Step by Step

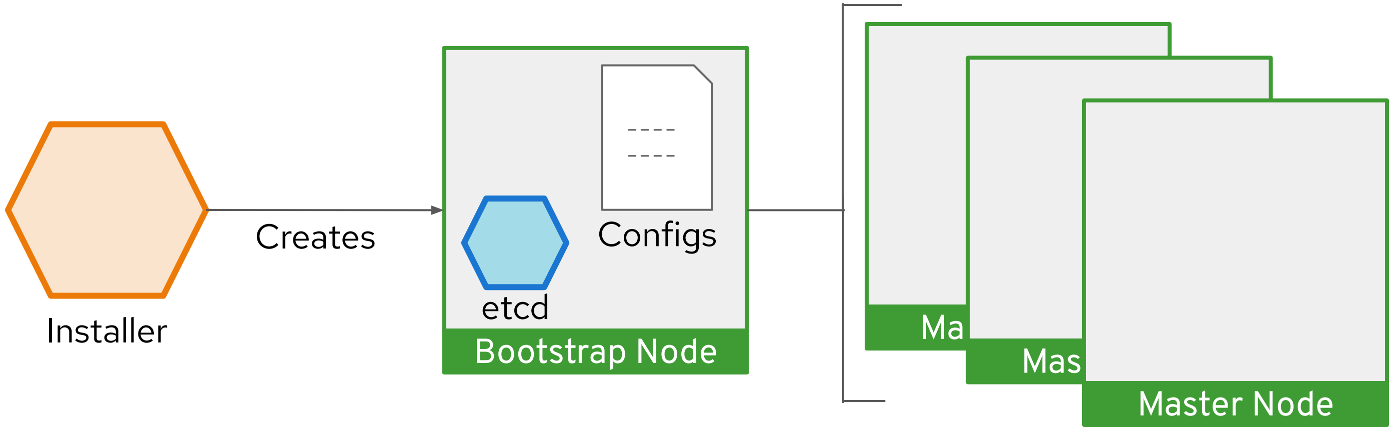




Bootstrapping process step by step:

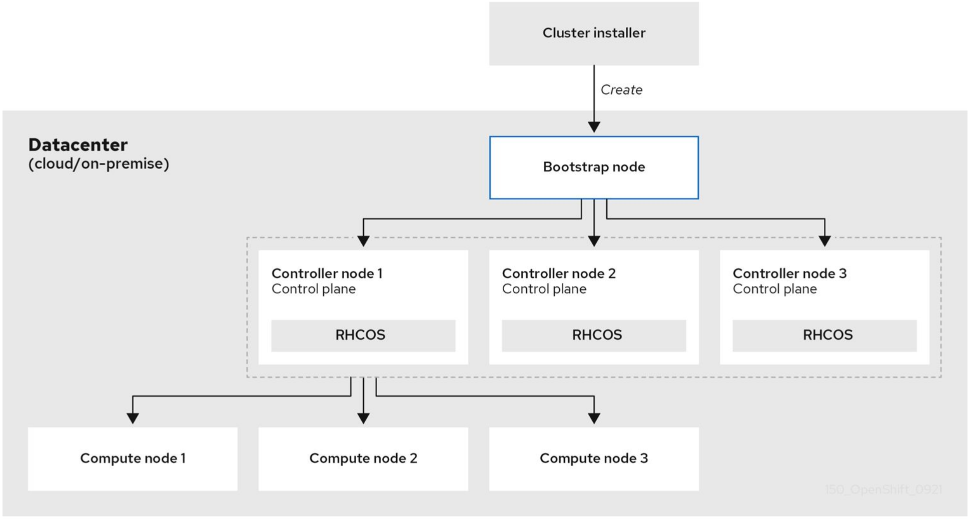
1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd

OpenShift Bootstrap Process: Step by Step

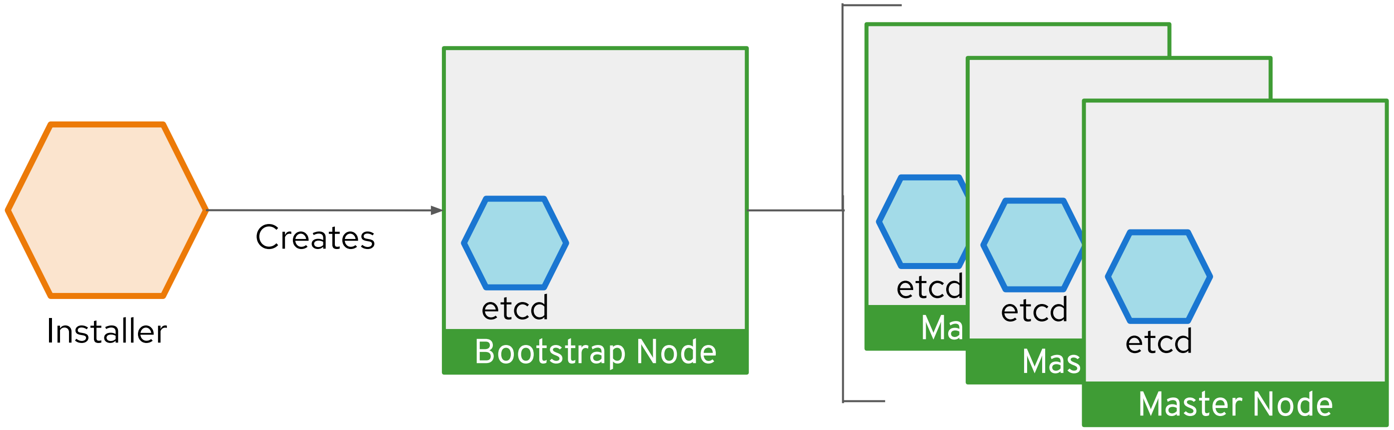


Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.



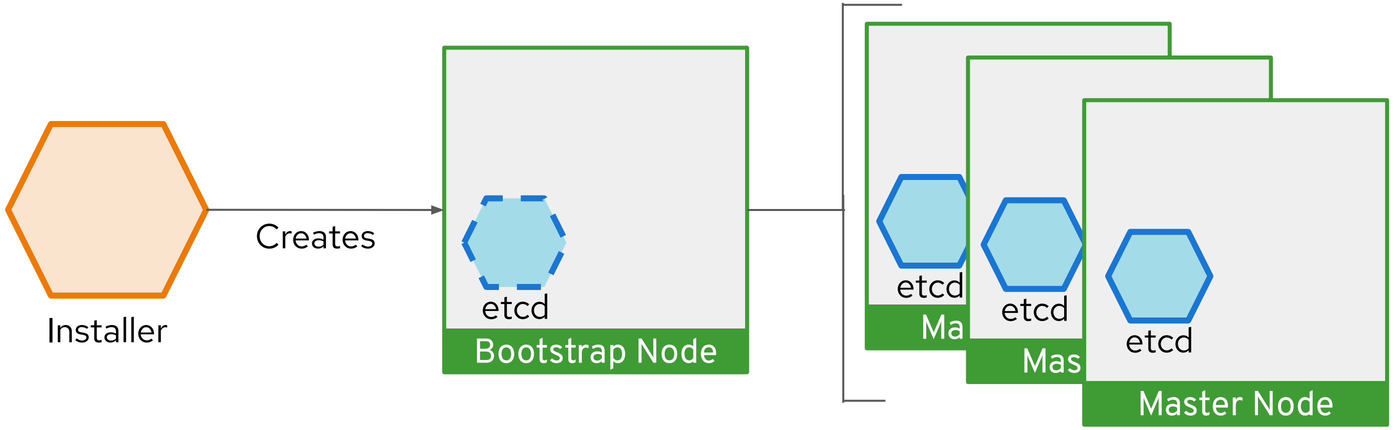
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 4 total instances.

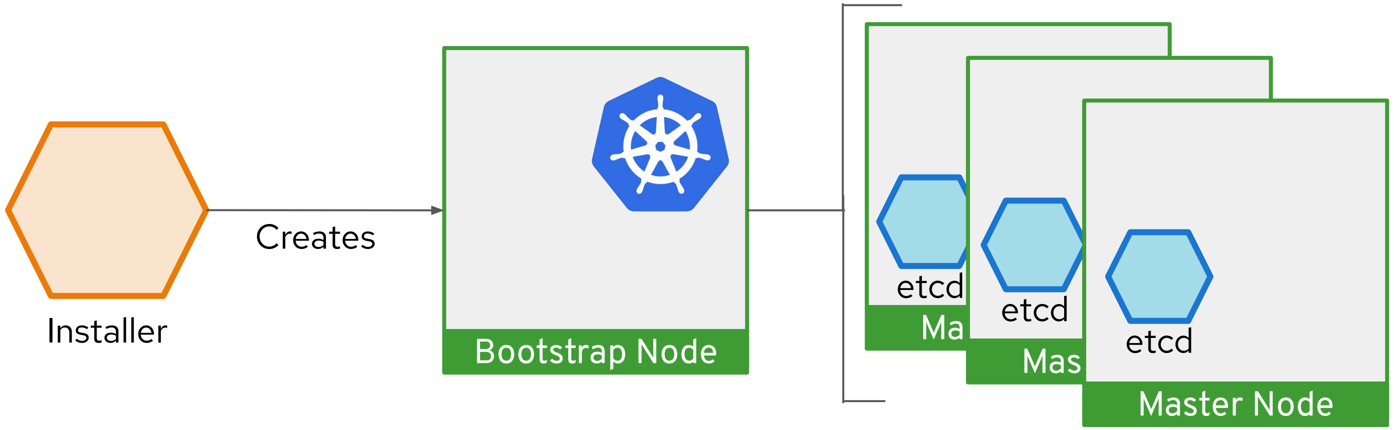
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 4 total instances.
4. The Etcd operator scales itself down off the bootstrap node, leaving the etcd instance count to 3

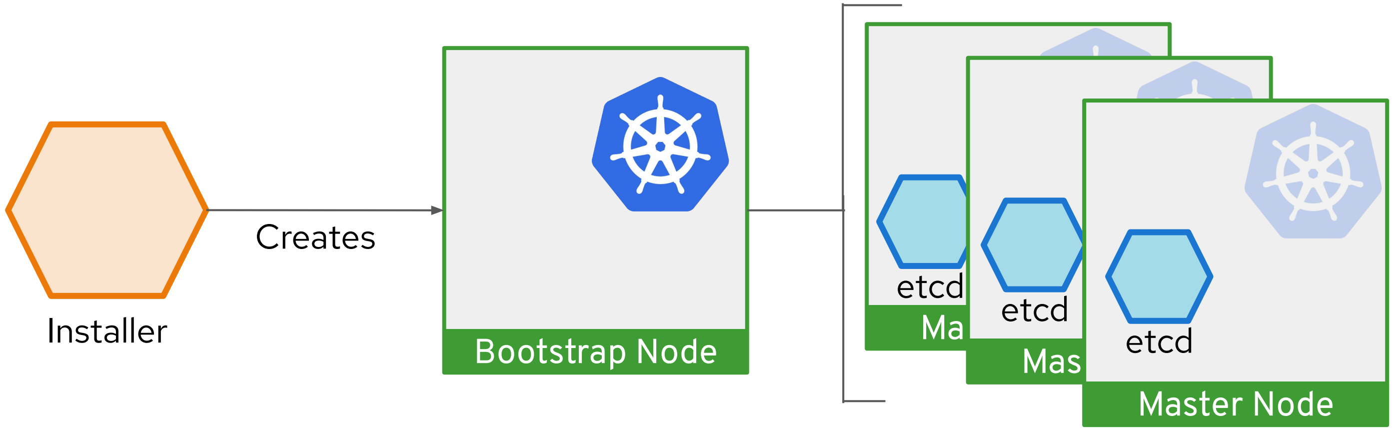
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.
4. The Etcd operator scales itself down off the bootstrap node, then scales back up to 3; all on the Masters
5. Bootstrap node starts a temporary Kubernetes control plane using the newly-created etcd cluster.

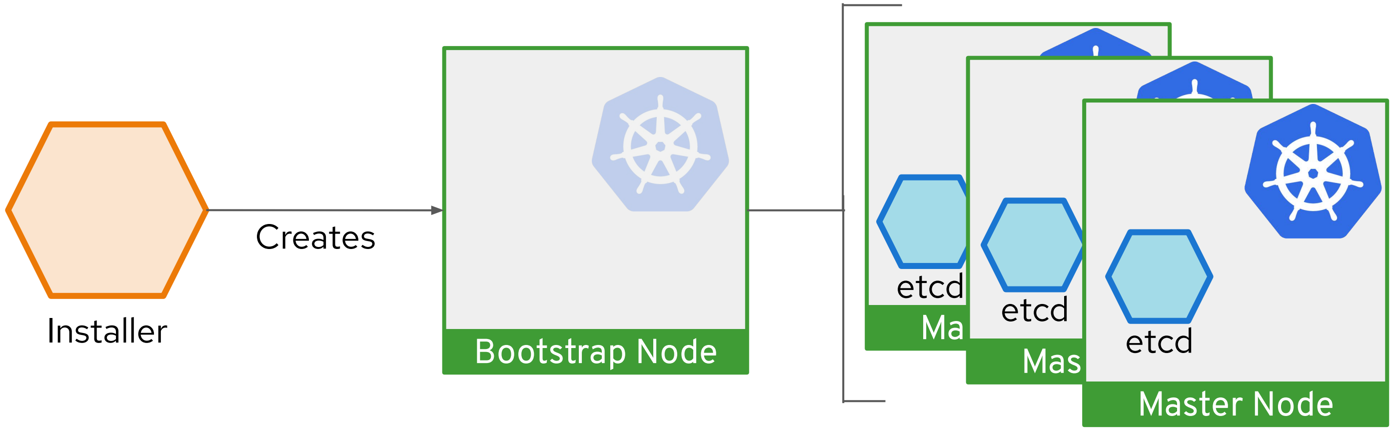
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.
4. The Etcd operator scales itself down off the bootstrap node, then scales back up to 3; all on the Masters
5. Bootstrap node starts a temporary Kubernetes control plane using the newly-created etcd cluster.
6. Temporary control plane schedules the production control plane to the master machines.

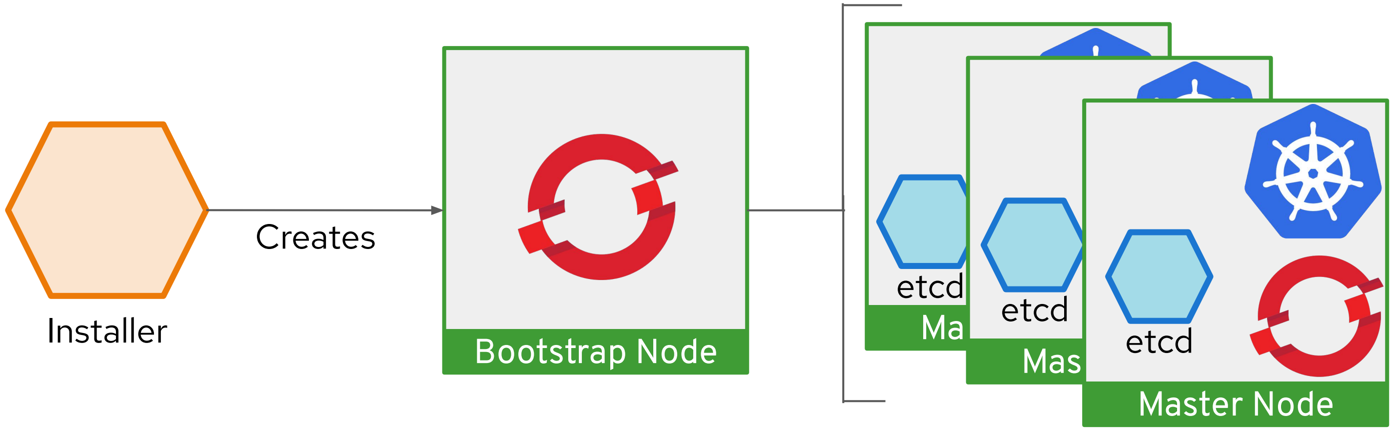
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.
4. The Etcd operator scales itself down off the bootstrap node, then scales back up to 3; all on the Masters
5. Bootstrap node starts a temporary Kubernetes control plane using the newly-created etcd cluster.
6. Temporary control plane schedules the production control plane to the master machines.
7. Temporary control plane shuts down, yielding to the production control plane.

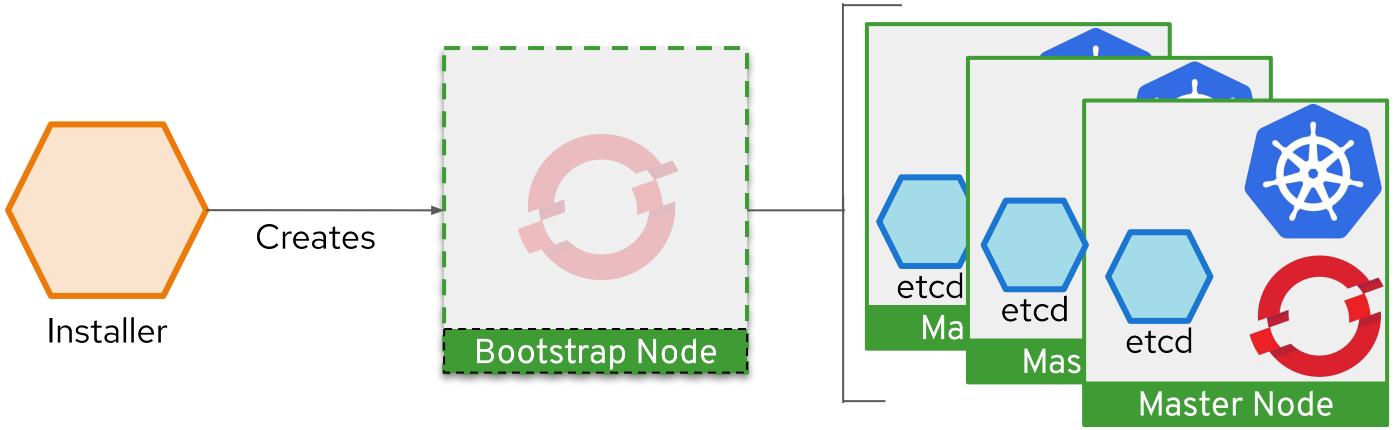
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.
4. The Etcd operator scales itself down off the bootstrap node, then scales back up to 3; all on the Masters
5. Bootstrap node starts a temporary Kubernetes control plane using the newly-created etcd cluster.
6. Temporary control plane schedules the production control plane to the master machines.
7. Temporary control plane shuts down, yielding to the production control plane.
8. Bootstrap node injects OpenShift-specific components into the newly formed control plane.

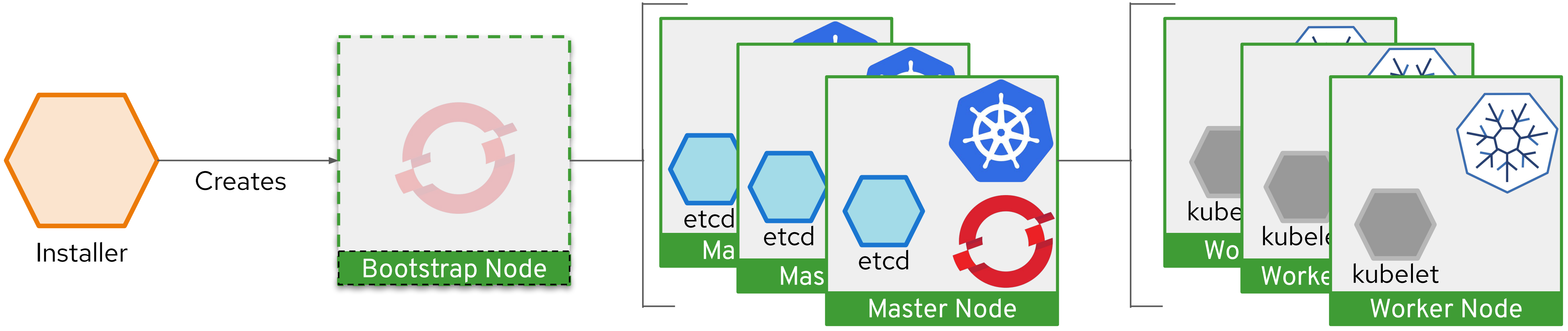
OpenShift Bootstrap Process: Step by Step



Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.
4. The Etcd operator scales itself down off the bootstrap node, then scales back up to 3; all on the Masters
5. Bootstrap node starts a temporary Kubernetes control plane using the newly-created etcd cluster.
6. Temporary control plane schedules the production control plane to the master machines.
7. Temporary control plane shuts down, yielding to the production control plane.
8. Bootstrap node injects OpenShift-specific components into the newly formed control plane.
9. Installer then tears down the bootstrap node or if user-provisioned, this needs to be performed by the administrator.

OpenShift Bootstrap Process: Step by Step



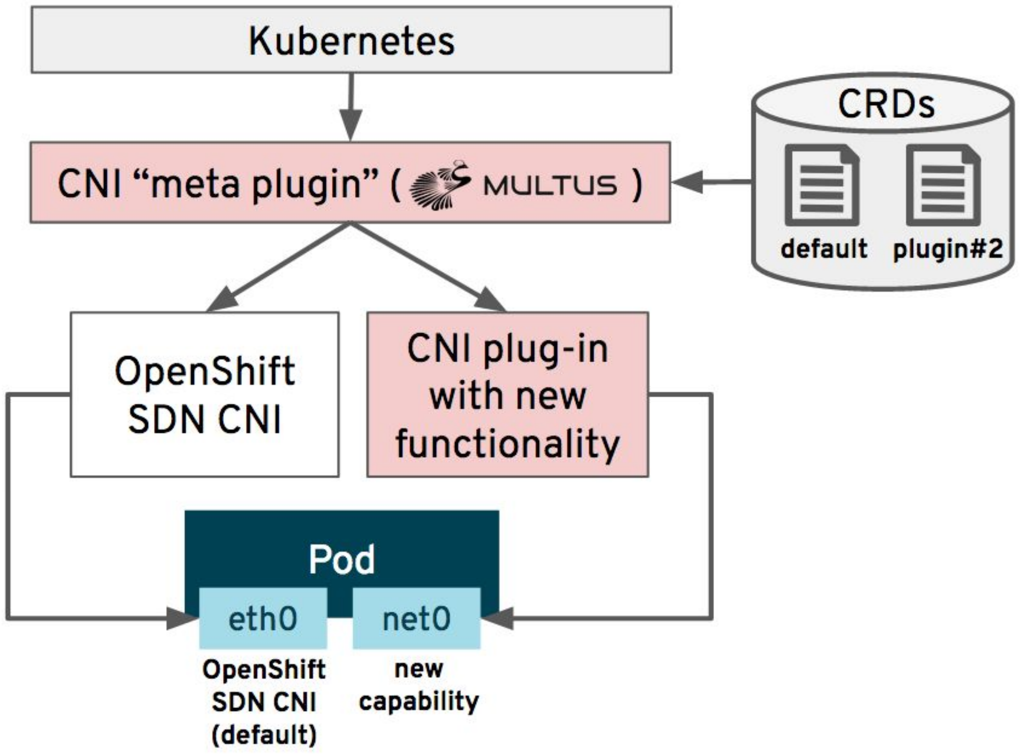
Bootstrapping process step by step:

1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.
4. The Etcd operator scales itself down off the bootstrap node, then scales back up to 3; all on the Masters
5. Bootstrap node starts a temporary Kubernetes control plane using the newly-created etcd cluster.
6. Temporary control plane schedules the production control plane to the master machines.
7. Temporary control plane shuts down, yielding to the production control plane.
8. Bootstrap node injects OpenShift-specific components into the newly formed control plane.
9. Installer then tears down the bootstrap node or if user-provisioned, this needs to be performed by the administrator.
10. Worker machines fetch remote resources from masters and finish booting.

Network

Virtual Machine Networking

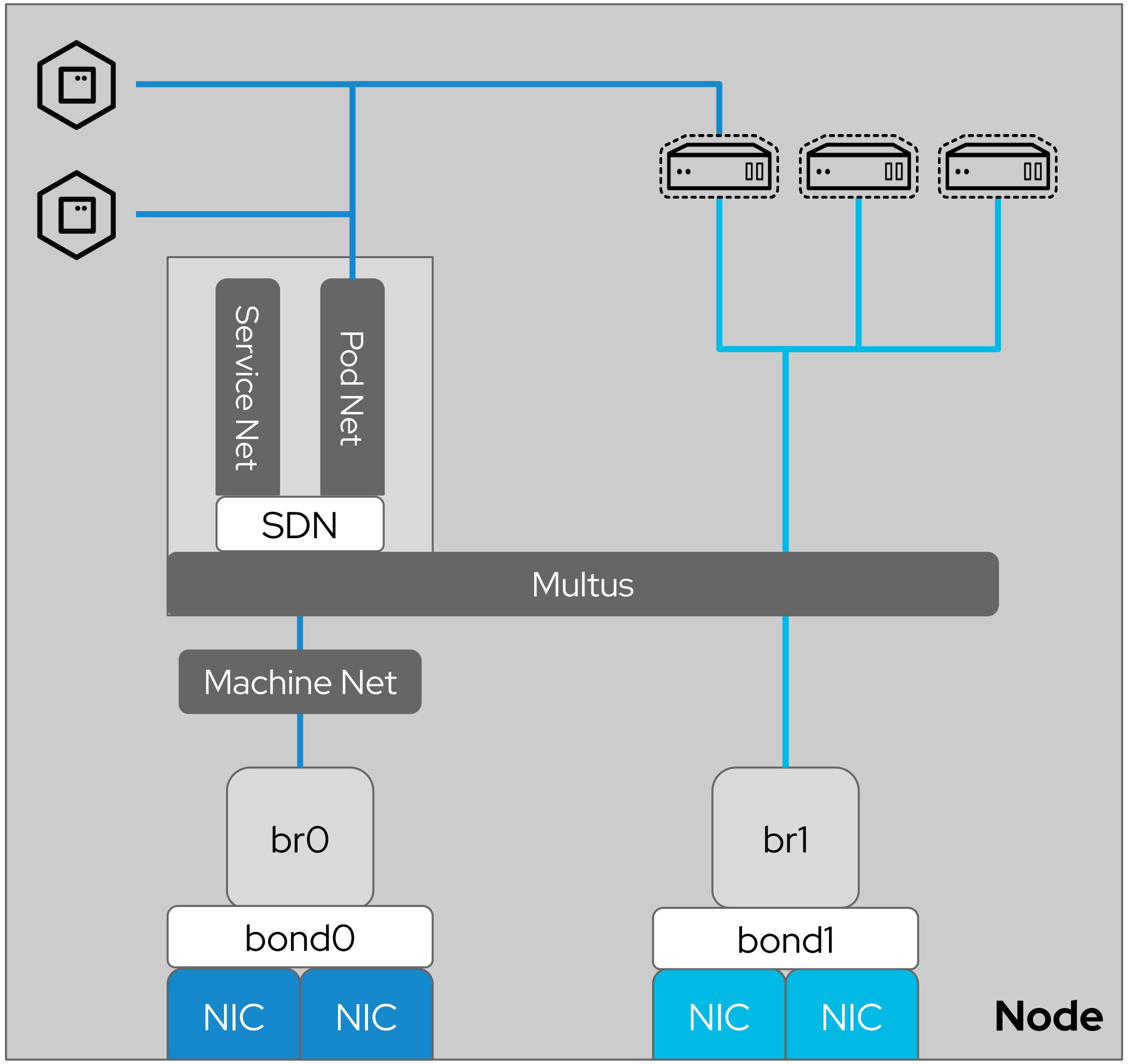
- Virtual machines optionally connect to the standard pod network
  - OpenShift SDN, OVNKubernetes
  - Partners, such as Calico, are also supported
- Additional network interfaces accessible via Multus:
  - Bridge, SR-IOV, OVN secondary networks
  - VLAN and other networks can be created at the host 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



Example host network configuration

- 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

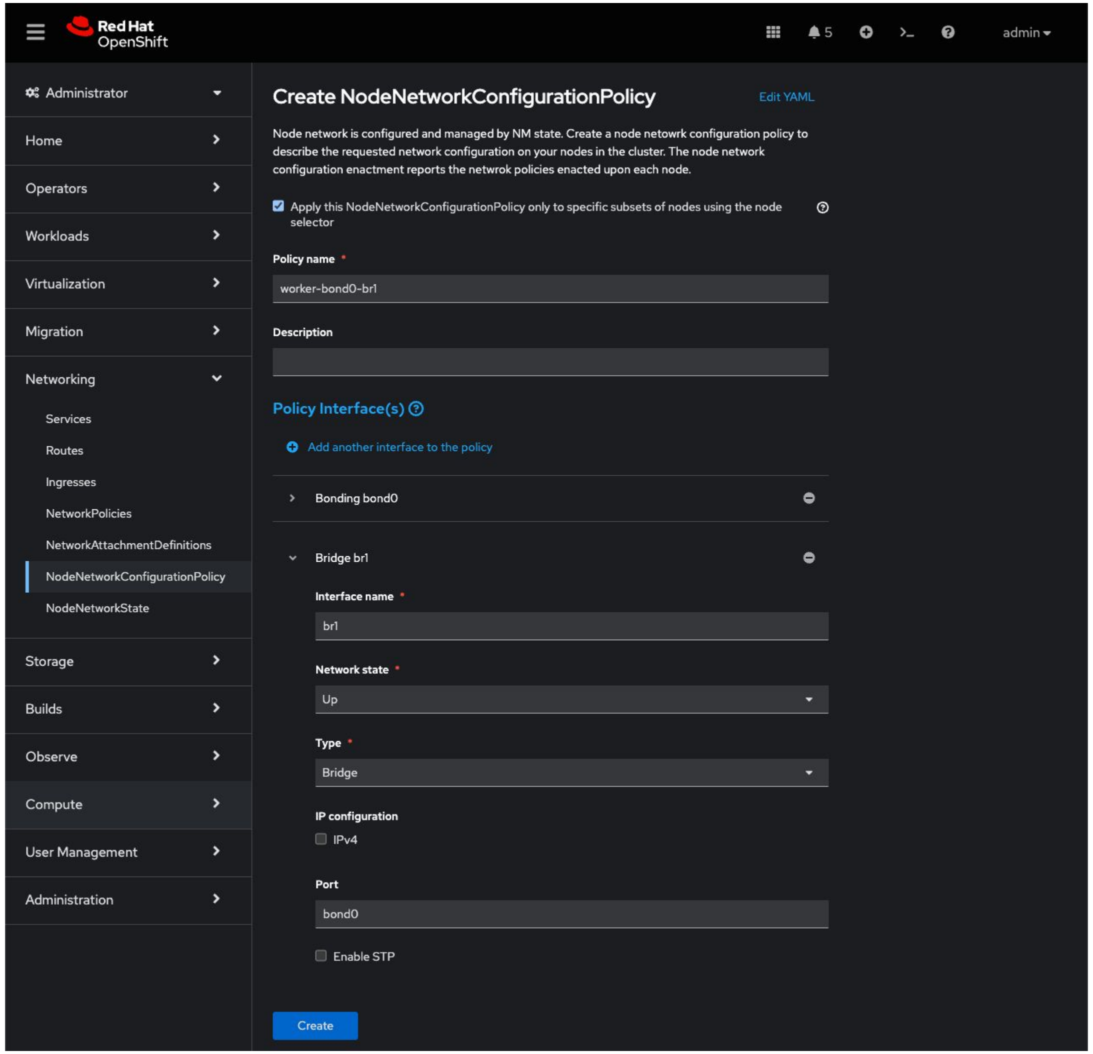
The following slides show an example of how this setup is configured



Red Hat

GUI-based host network configuration

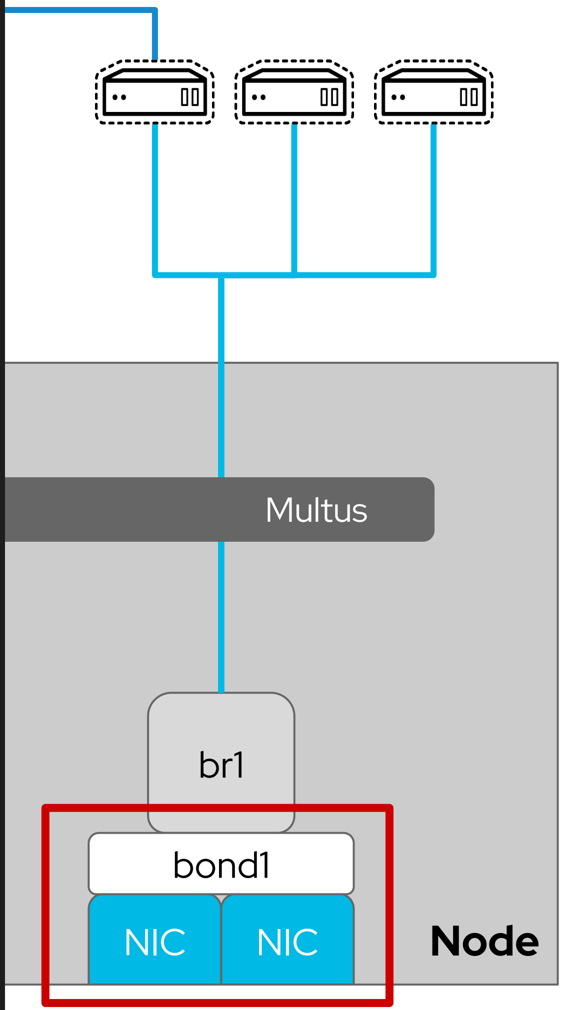
- Apply NMstate configuration using a form in the OpenShift admin console
- Create and configure
  - Ethernet interface IP (static, DHCP)
  - Bonds - mode 1-6 bonds with options, including IP configuration
  - Linux bridge configuration utilizing ethernet and/or bonds for "uplinks"
- Specify node selectors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



Host bond configuration

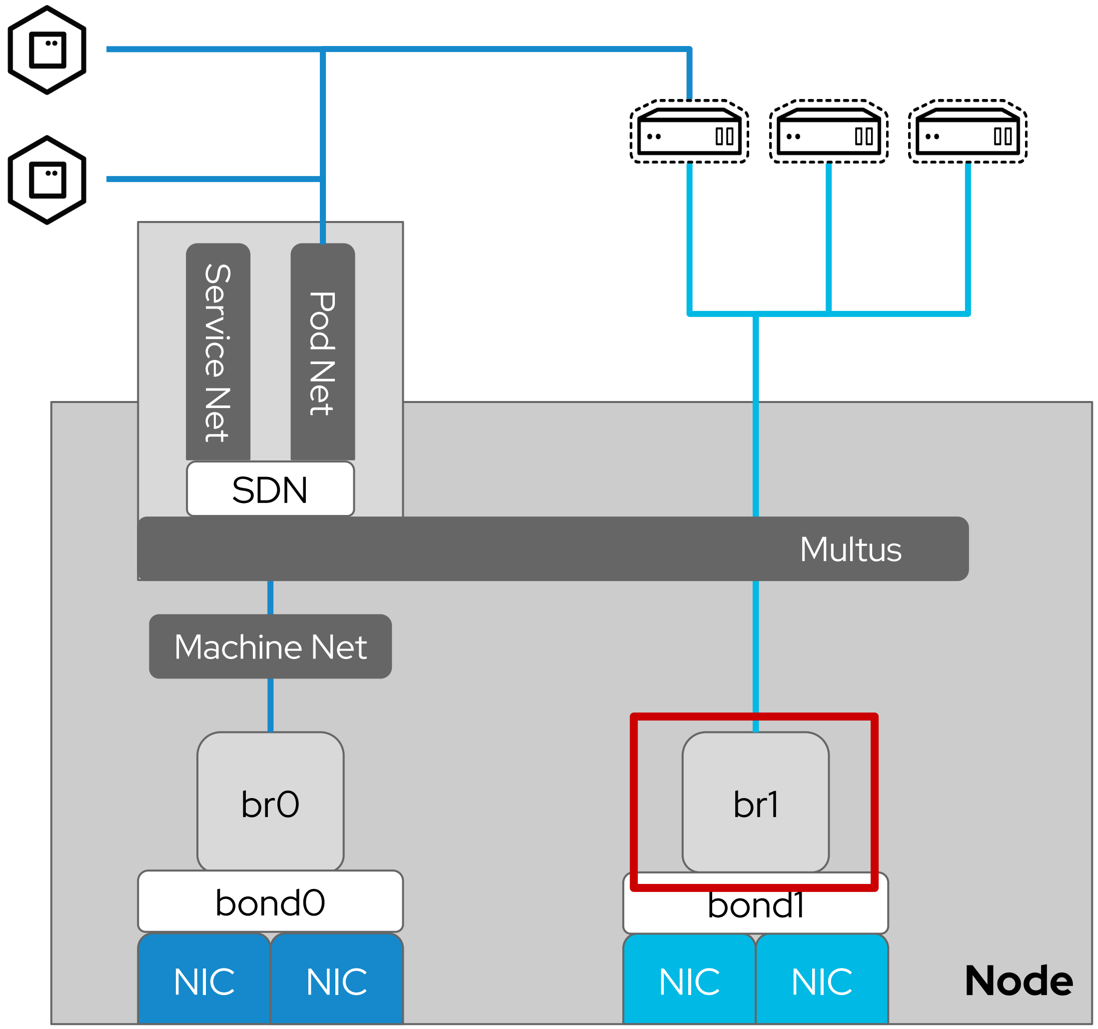
- NodeNetworkConfiguration-Policy (NNCP)
  - Nmstate operator CRD
  - Configure host network using declarative language
- Applies to all nodes specified in the `nodeSelector`, including newly added nodes automatically
- Update or add new NNCPs for additional host configs

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



Host bridge configuration

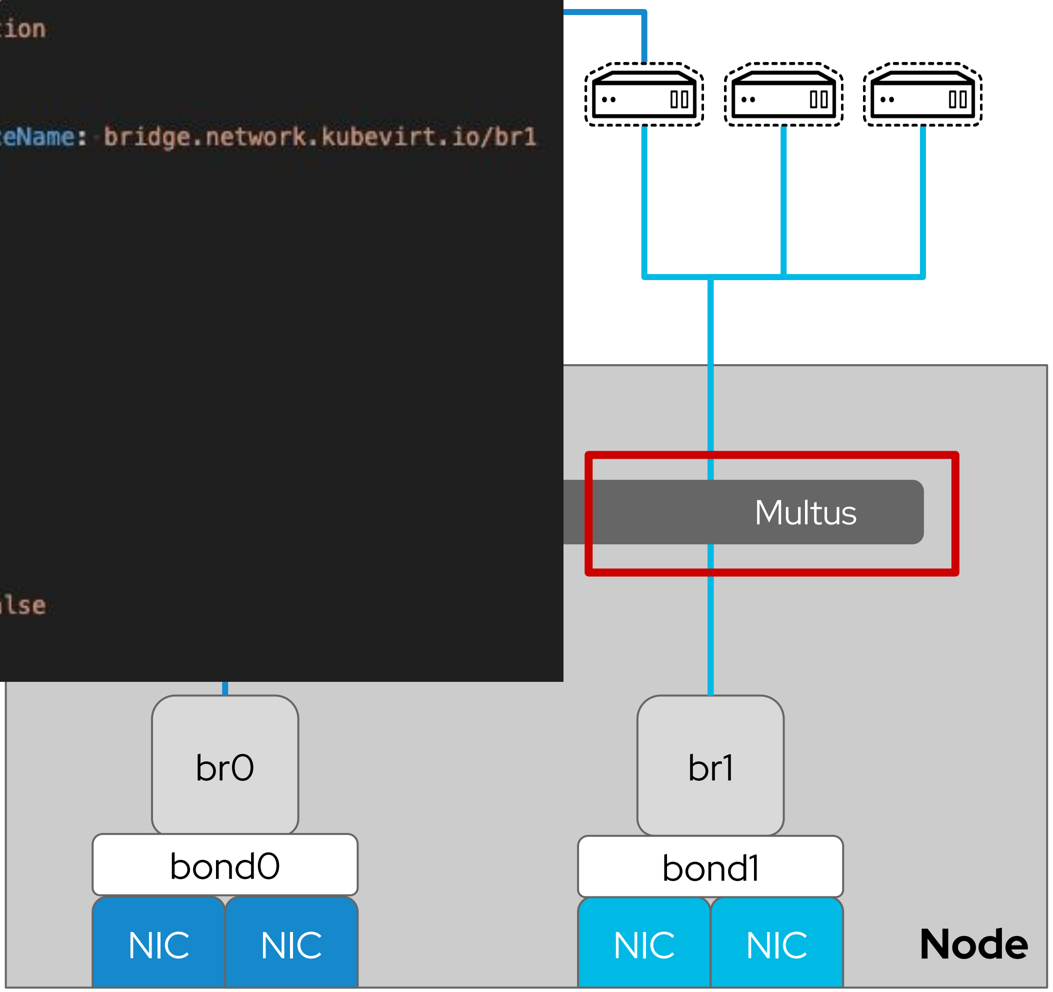
```yaml
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: worker-bond1-br1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  desiredState:
    interfaces:
      - name: br1
        description: br1 with bond1
        type: linux-bridge
        state: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



Network Attachment Definition configuration

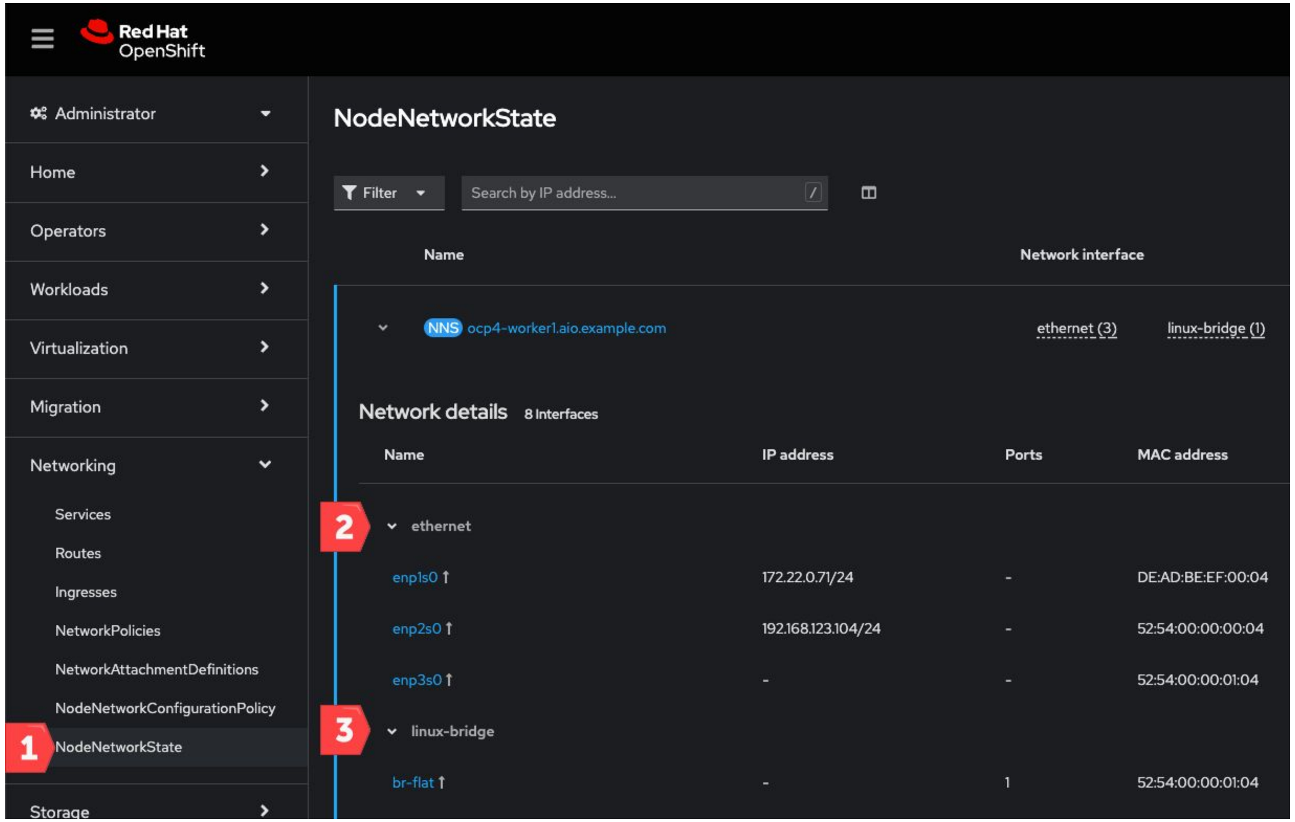
- `net-attach-def` configures multus to allow the VM to access an underlying resource
  - Optionally define VLAN tags
- Limited to the namespace it’s created in
  - Except the `default` namespace, which is available to all

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
  name: vlan-93
  namespace: default
spec:
  config: >-
    {
      "name":"vlan-93"
      "type":"cnv-bridge",
      "cniVersion":"0.3.1",
      "bridge":"br1",
      "vlan":93,"
      macspoofchk":true,
      "ipam":{},
      "preserveDefaultVlan":false
    }
```



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



Connecting Pods to networks

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

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

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


Red Hat

Connecting VMs to networks

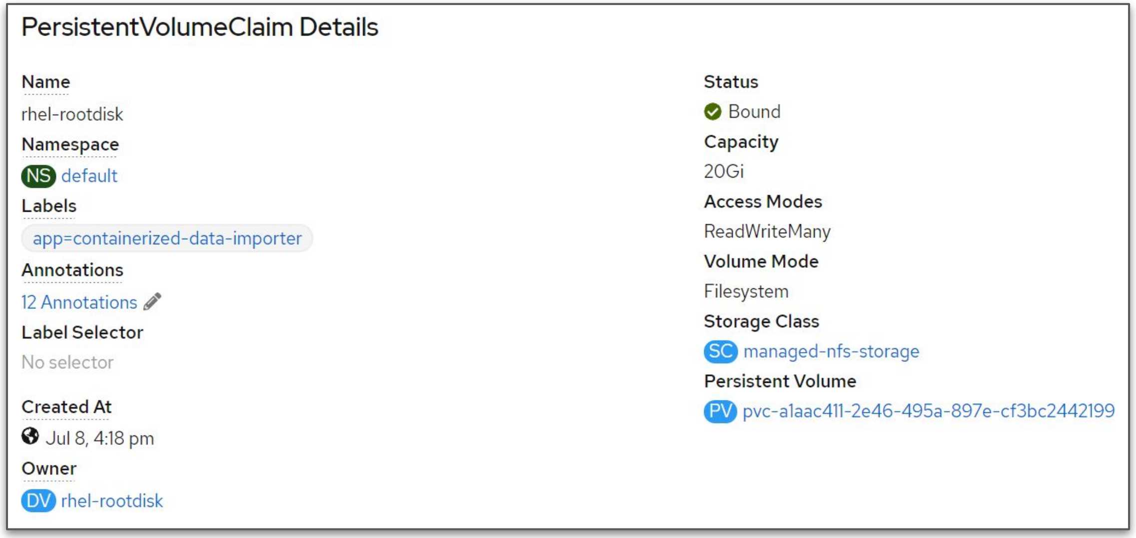
- Virtual machine interfaces describe NICs attached to the VM
  - `spec.domain.devices.interfaces`
  - Model: virtio, e1000, pcnet, rtl8139, etc.
  - Type: masquerade, bridge
  - MAC address: customize the MAC
- The networks definition describes the connection type
  - `spec.networks`
  - Pod = default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

```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

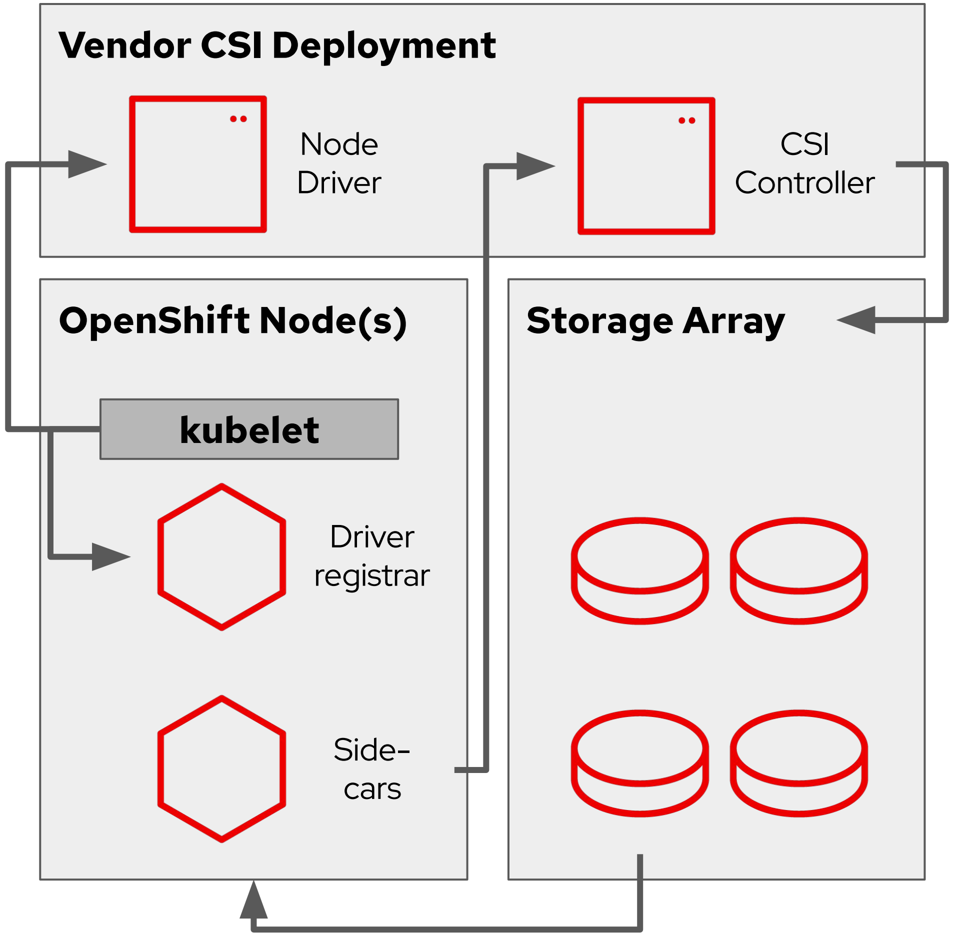
- OpenShift Virtualization uses the Kubernetes PersistentVolume (PV) paradigm
- PVs can be backed by
  - CSI drivers, including partners and ODF
  - Local storage using host path provisioner / logical volume Operator
- Use dynamically or statically provisioned PVs
- RWX is ***required*** for live migration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



Red Hat

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

```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'
```


Red Hat

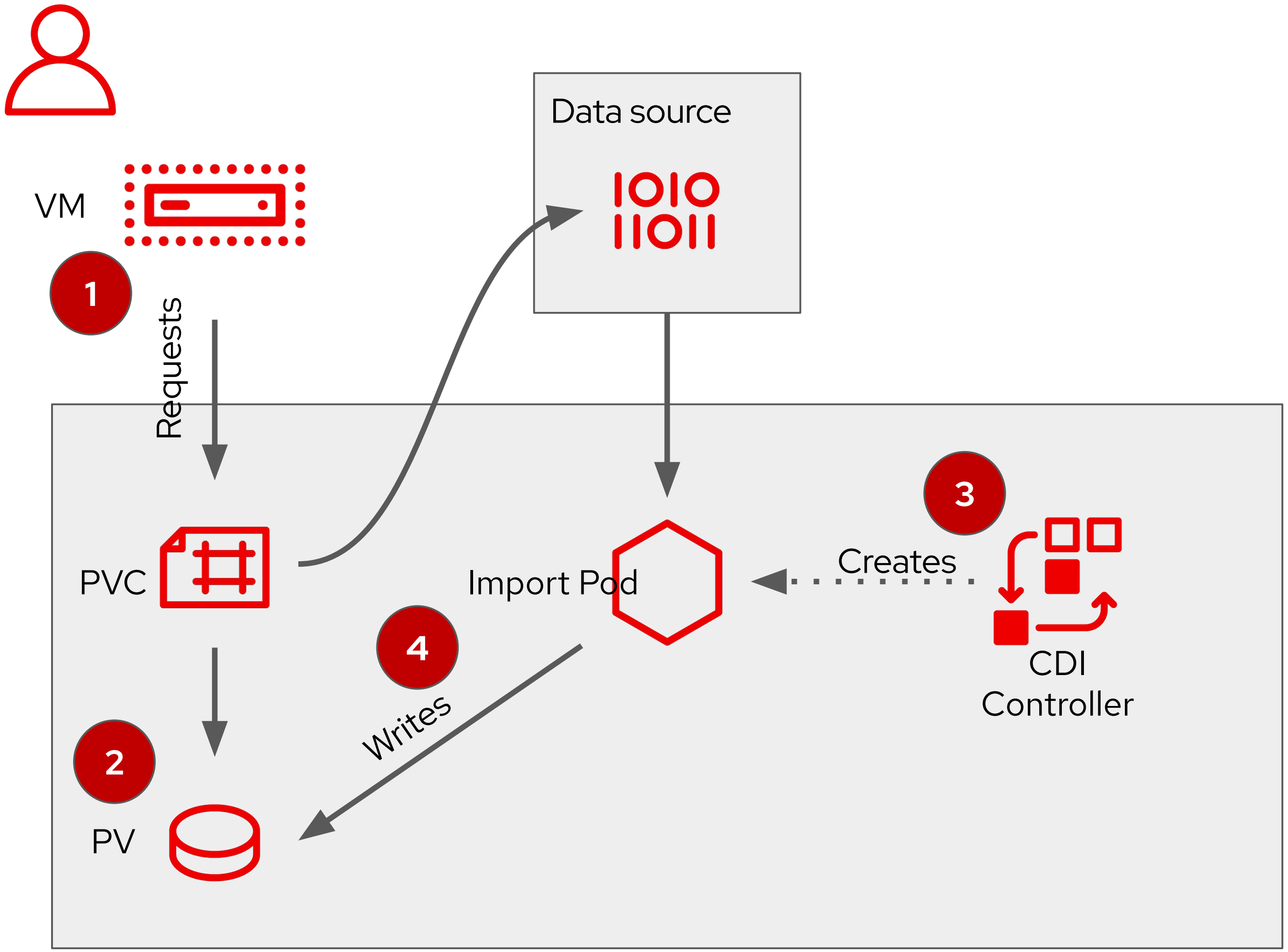
Storage Profiles

- 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

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: hostpath-provisioner
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce
    volumeMode:
      Filesystem
```

```yaml
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: rhel8-vm-disk
spec:
  storage:
    resources:
      requests:
        storage: 30Gi
    storageClassName: hostpath-provisioner
  source:
    pvc:
      name: rhel8-cloud-image
      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

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

Helper disks

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

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

Name	Source	Size	Interface	Storage Class
cloudinitdisk	Other	-	VirtIO	-

Comparing with traditional virtualization platforms

Live Migration

- Live migration moves a virtual machine from one node to another in the OpenShift cluster
- Can be triggered via GUI, CLI, API, or automatically
- **RWX storage is required**
- 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

Red Hat

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

Red Hat

Terminology comparison

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

Runtime awareness

Deploy and configure

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

Compute configuration

- VM nodes should be physical with CPU virtualization technology enabled in the BIOS
  - Nested virtualization *works*, but ***is not supported***
  - Emulation *works*, but ***is not supported*** (and is extremely slow)
- Node labeler detects CPU type and labels nodes for compatibility and scheduling
- Configure overcommitment using native OpenShift functionality - Cluster Resource Override Operator
  - Optionally, customize the 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

Red Hat

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=PLaR6Rq6Z4IqeQeTosfoFzTyE_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

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat