Red Hat OpenShift

OpenShift Virtualization

Technical Deep Dive

Alfred Bach
PSA Red Hat
Field Partner and Learning Team

Red Hat

Agenda

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

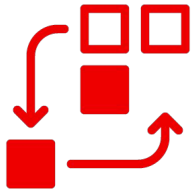
Install OpenShift Container Platform (OCP)

Red Hat

OpenShift 4.16 Supported Providers

Installation Experiences

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



Full Stack Automation

Installer Provisioned Infrastructure

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

Pre-existing Infrastructure

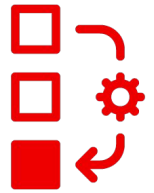
User Provisioned Infrastructure

- 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

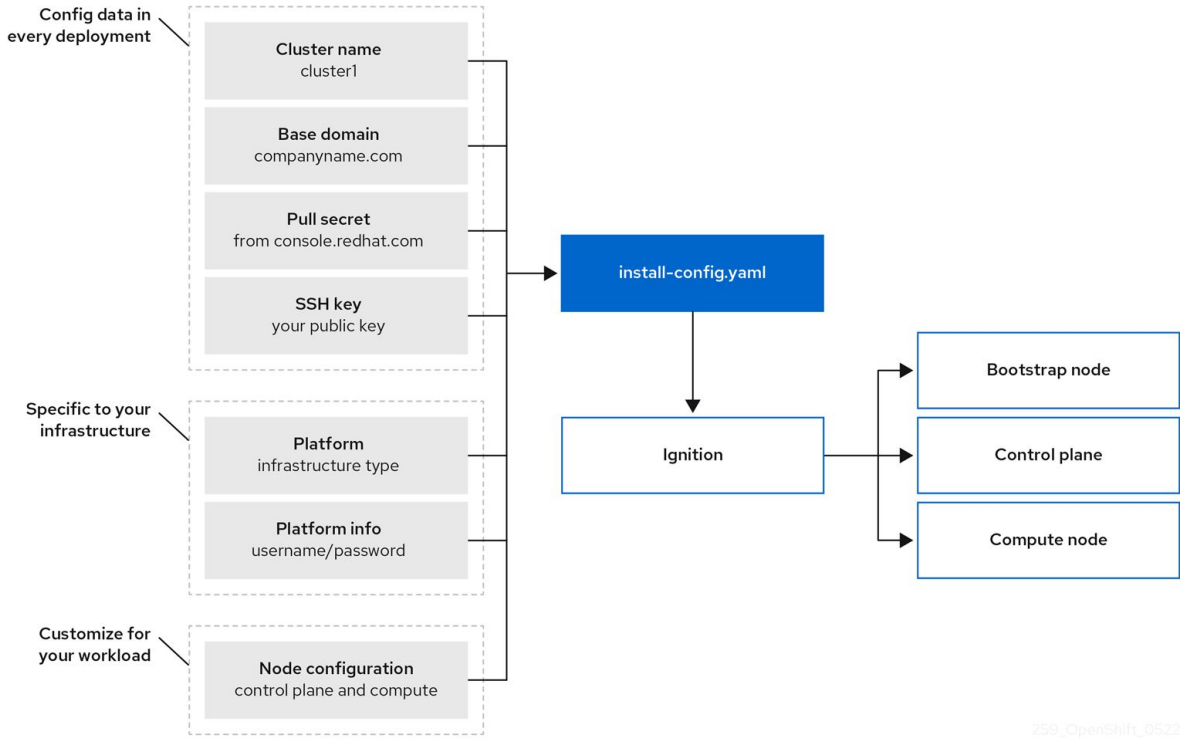
Red Hat

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)

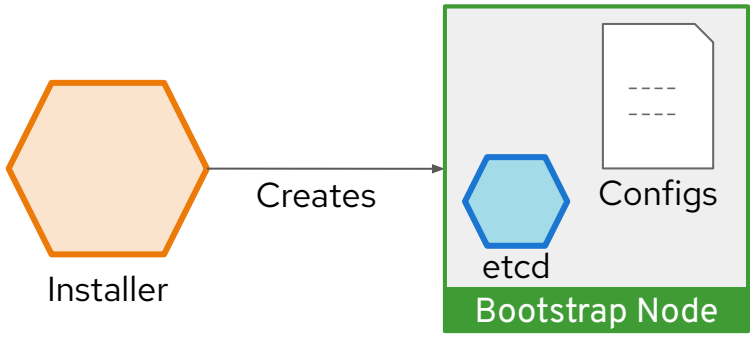
OpenShift Bootstrap Process: Self-Managed Kubernetes

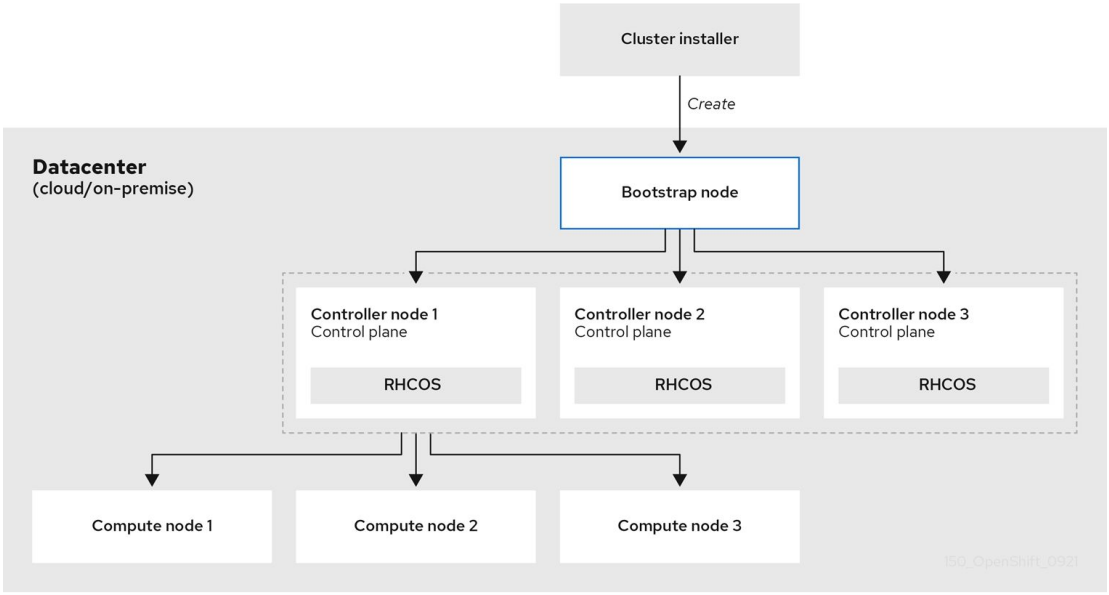
How to boot a self-managed cluster:

- 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

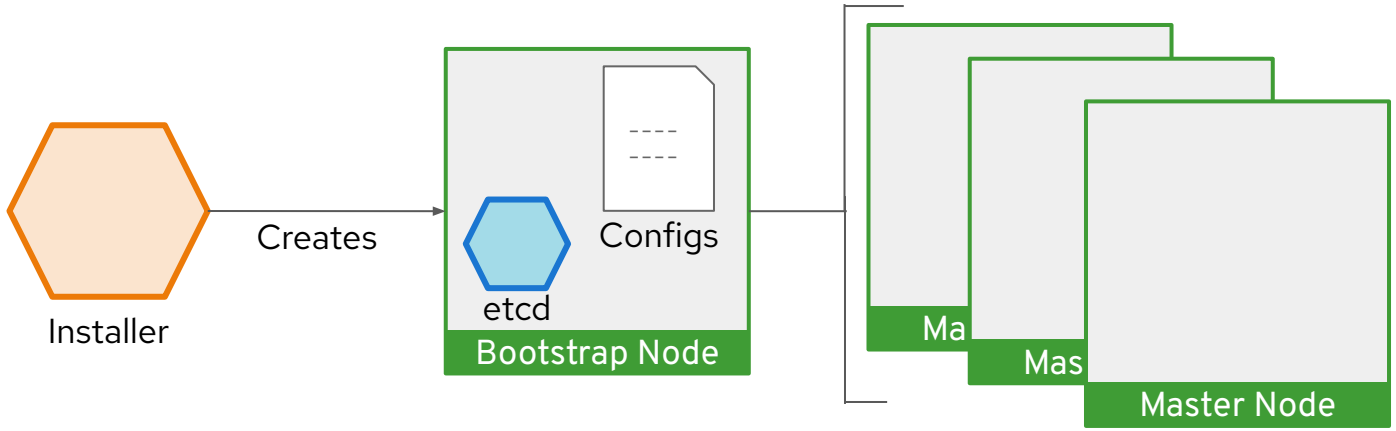




Bootstrapping process step by step:

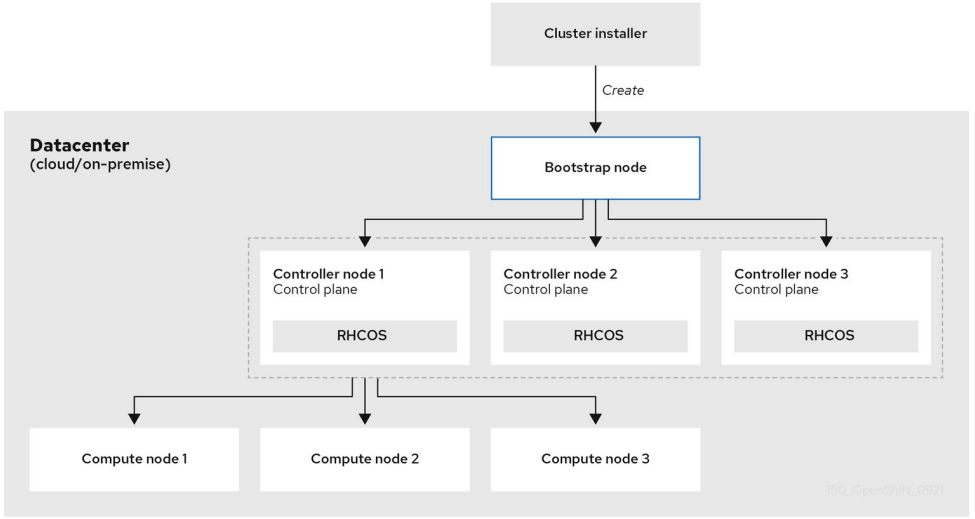
1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd

OpenShift Bootstrap Process: Step by Step

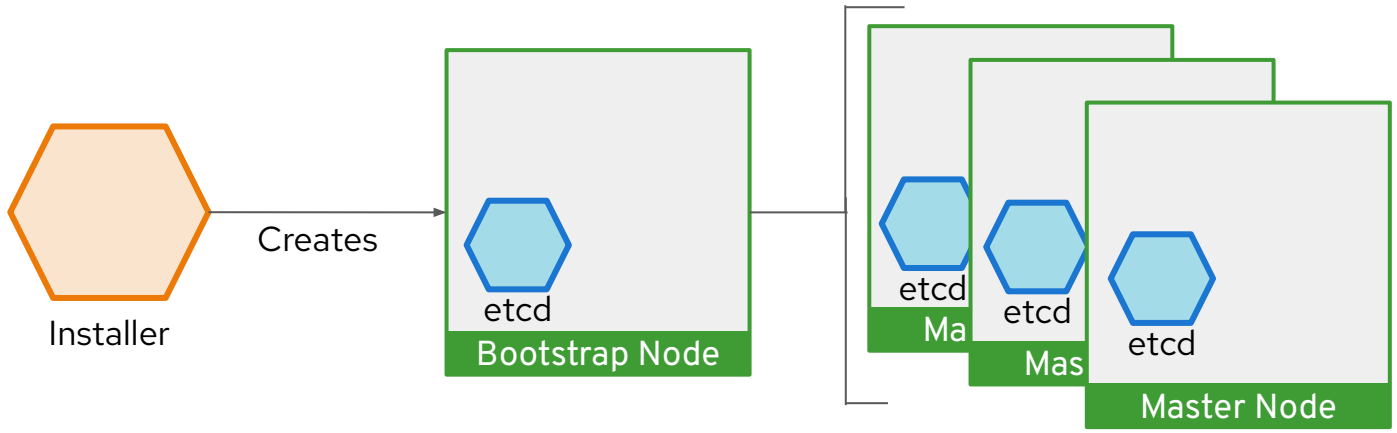


Bootstrapping process step by step:

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.



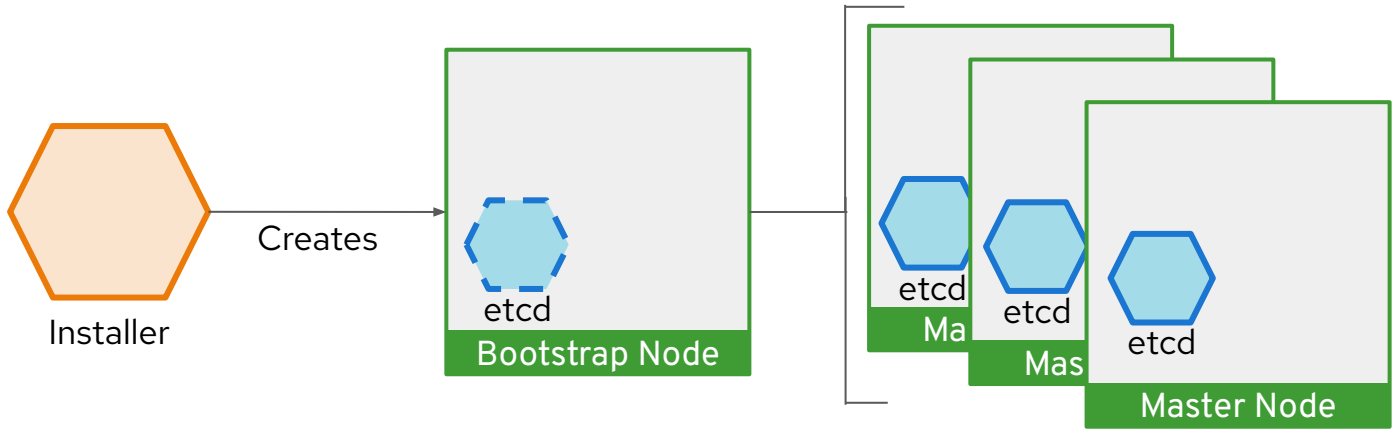
OpenShift Bootstrap Process: Step by Step



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.

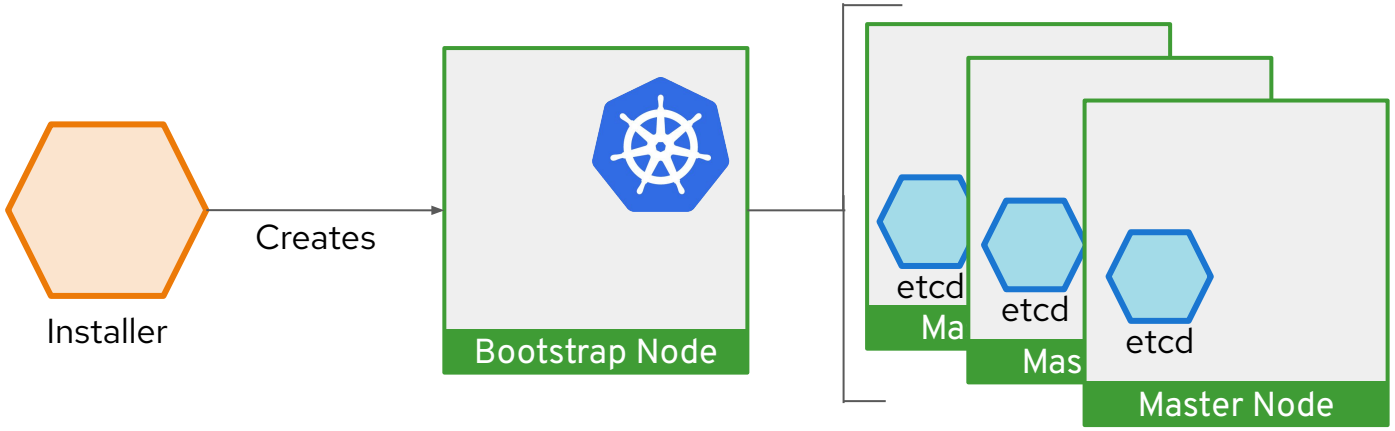
OpenShift Bootstrap Process: Step by Step



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

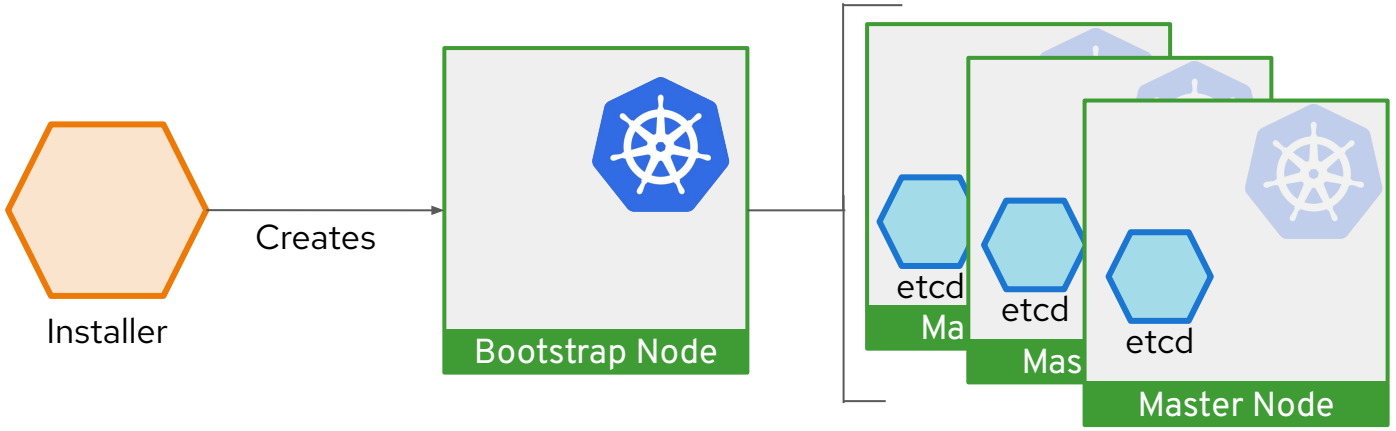
OpenShift Bootstrap Process: Step by Step



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.

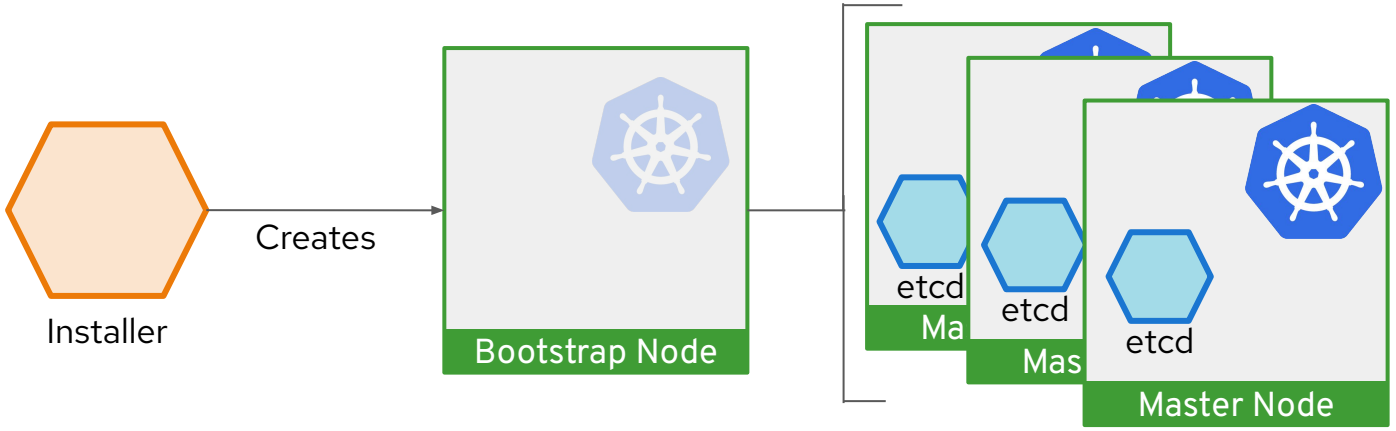
OpenShift Bootstrap Process: Step by Step



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.

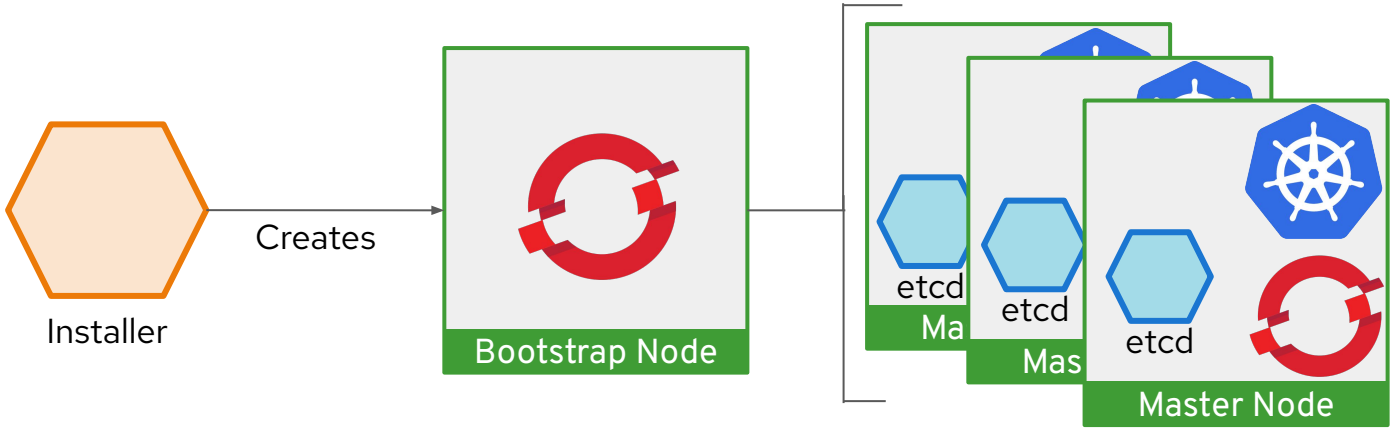
OpenShift Bootstrap Process: Step by Step



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.

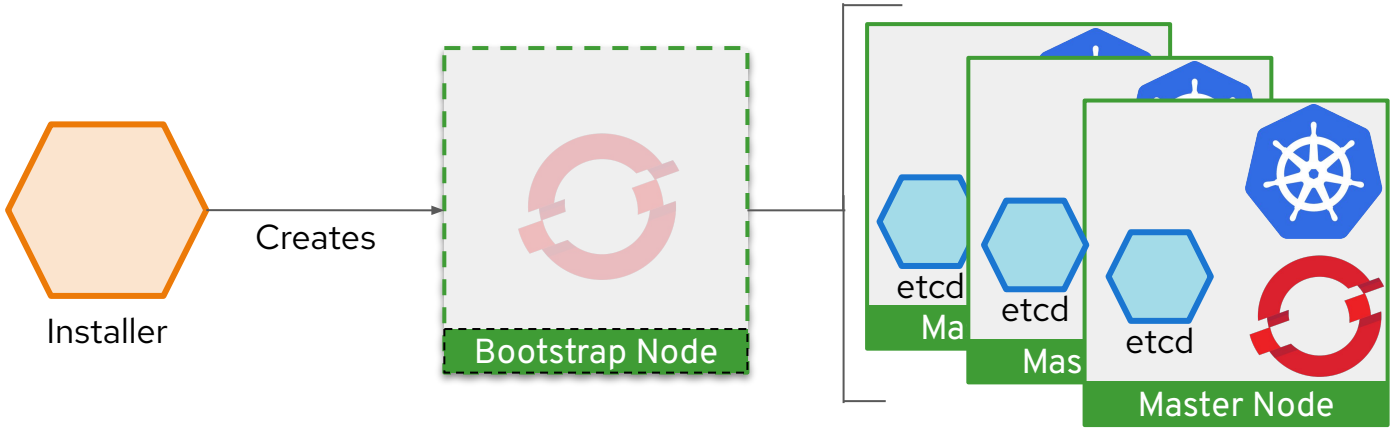
OpenShift Bootstrap Process: Step by Step



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.

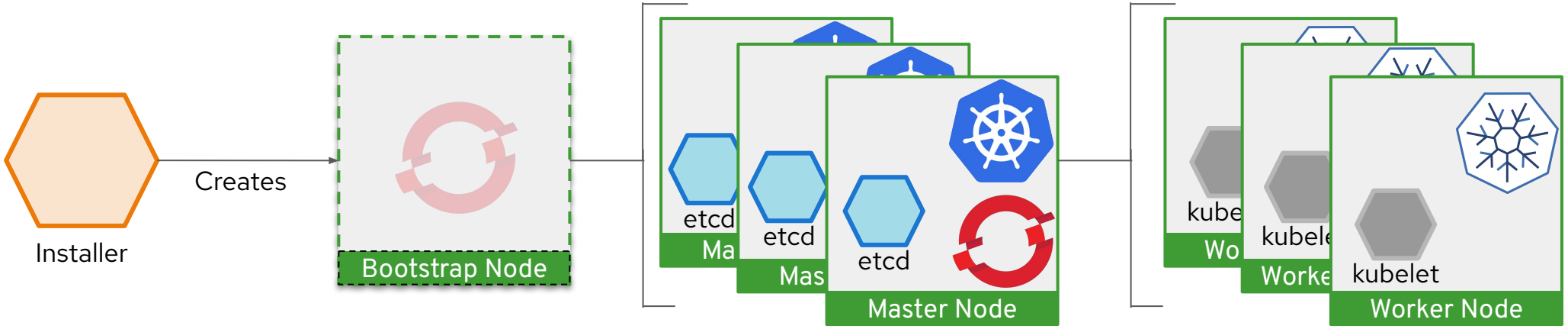
OpenShift Bootstrap Process: Step by Step



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:

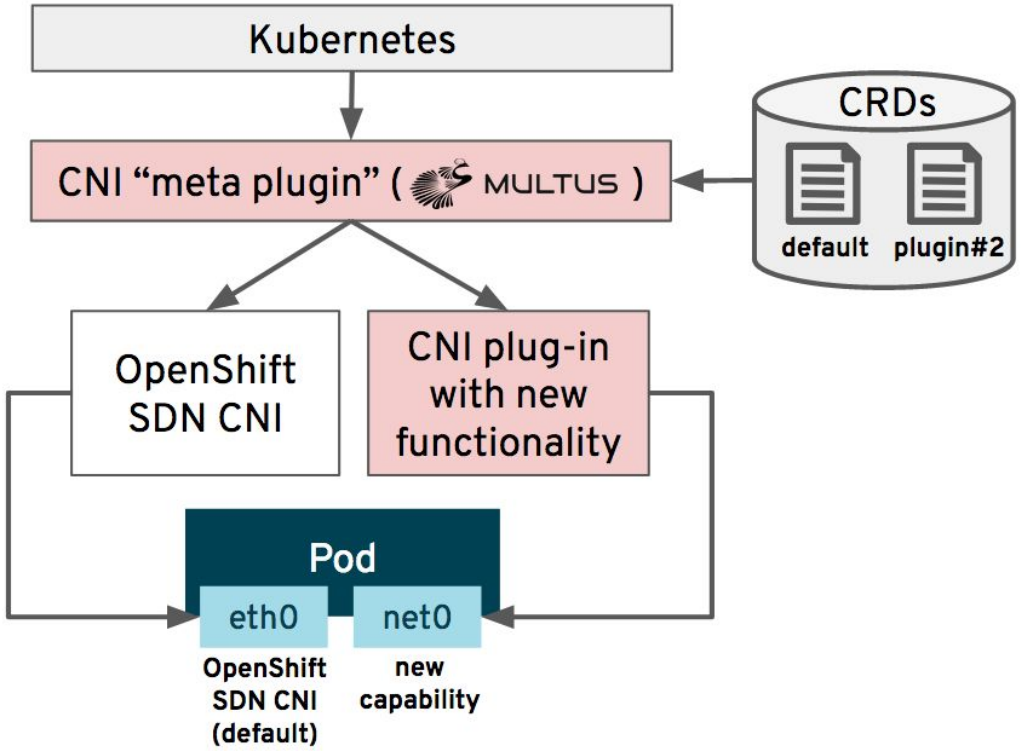
1. Bootstrap machine boots and starts hosting the remote resources required for master machines to boot. Runs one instance of etcd
2. Master machines fetch the remote resources from the bootstrap machine and finish booting.
3. Master machines use the bootstrap node to scale the etcd cluster to 3 instances.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

Red Hat

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

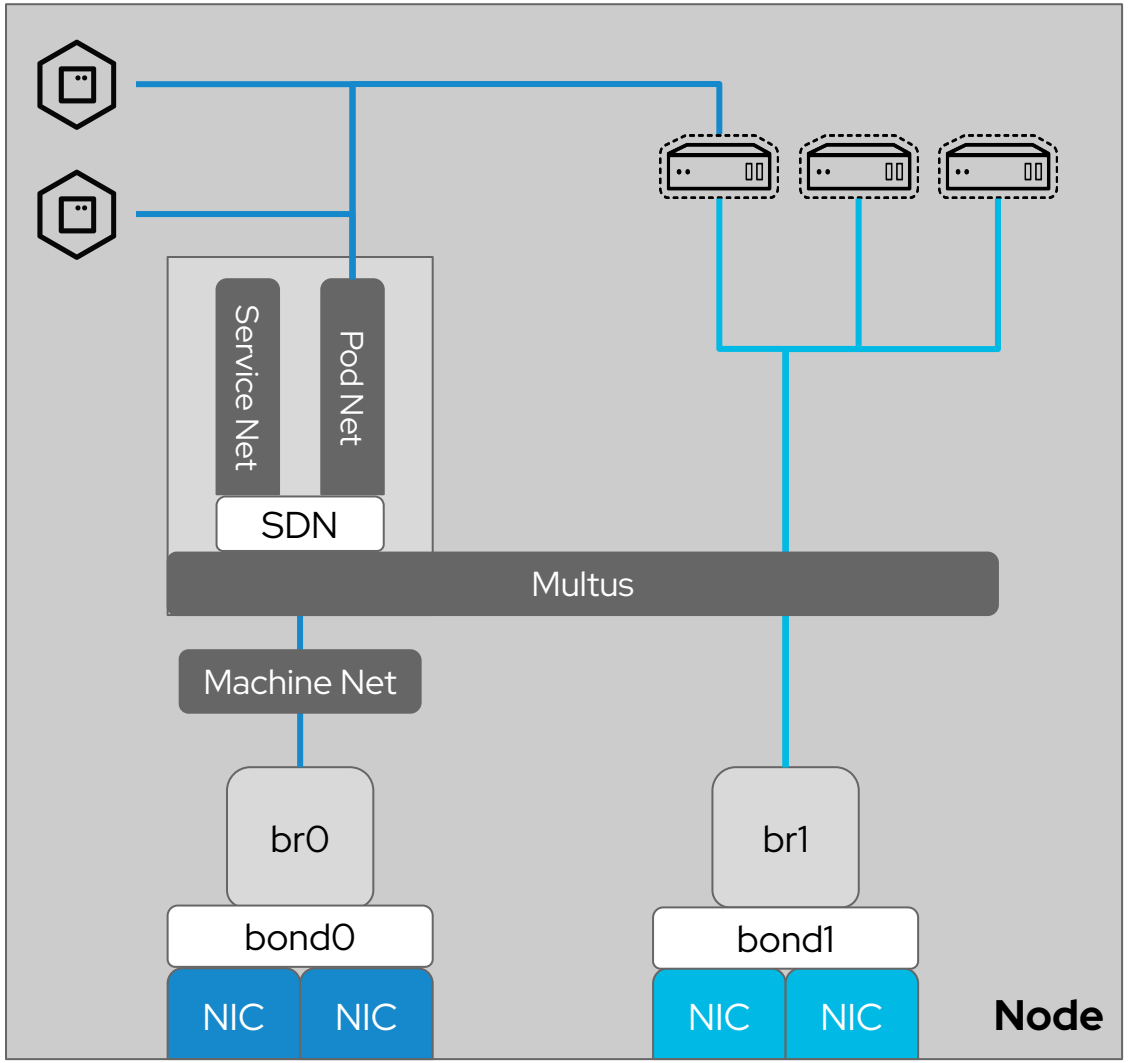


Red Hat

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

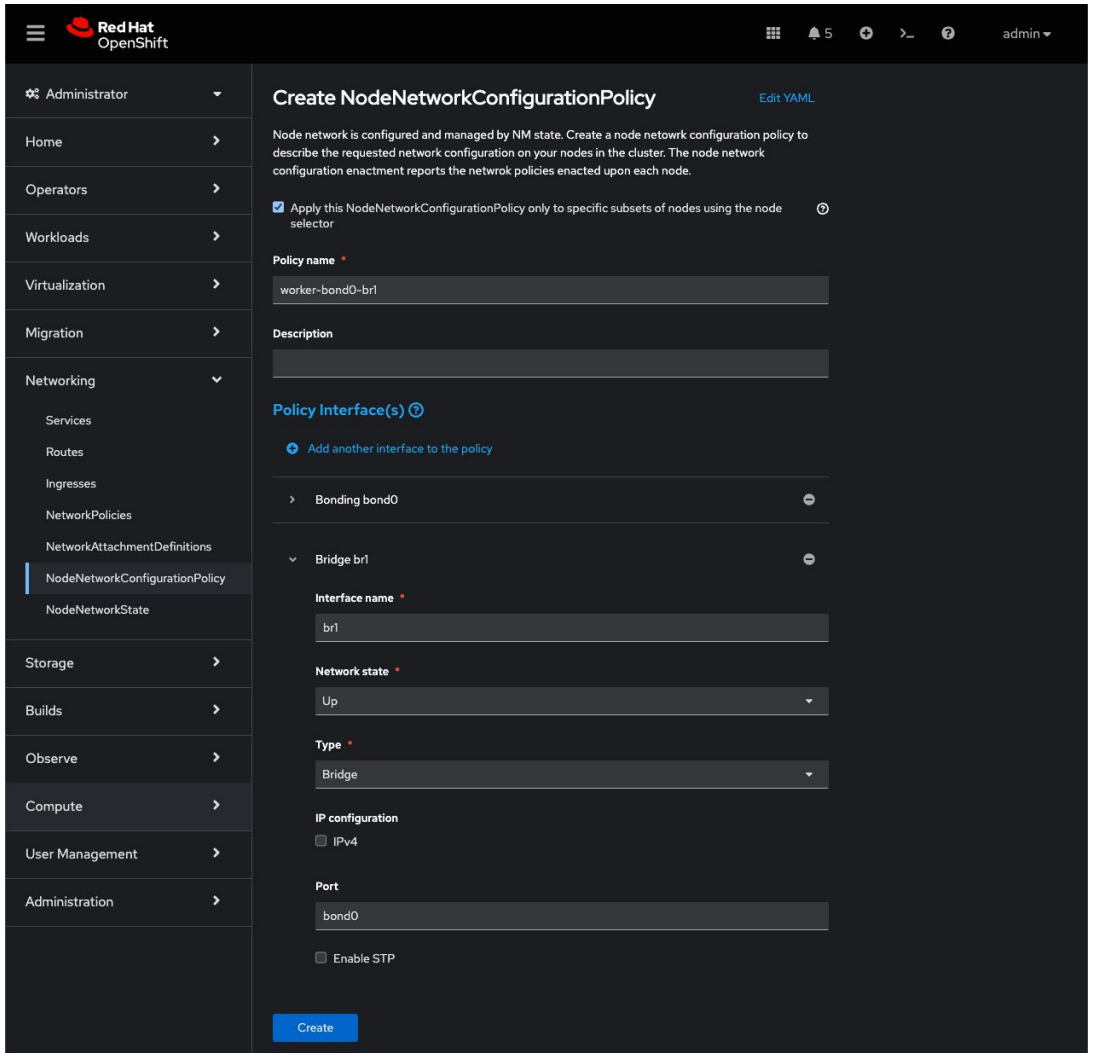
The following slides show an example of how this setup is configured



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

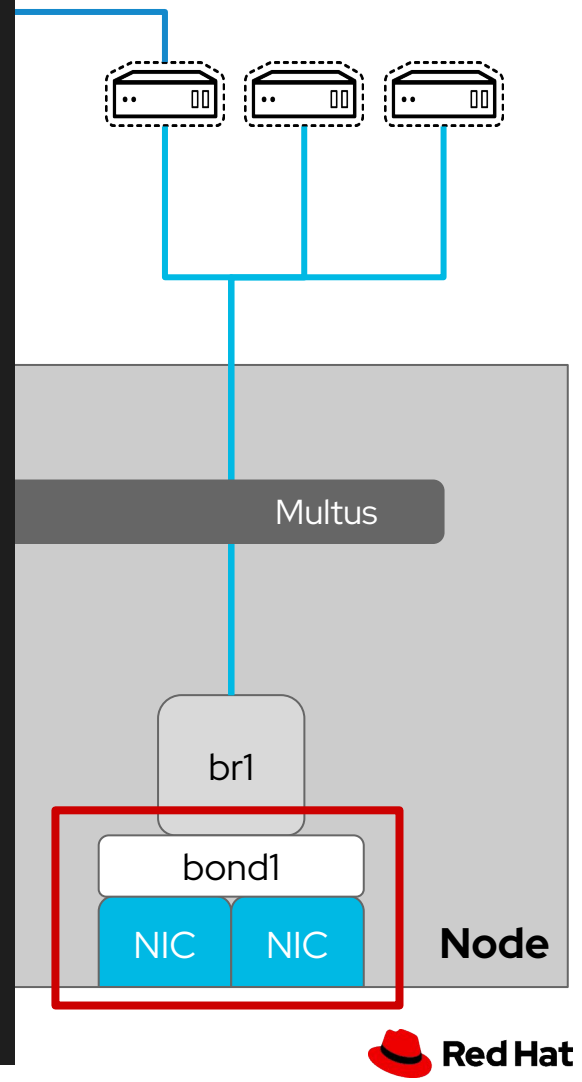


Red Hat

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

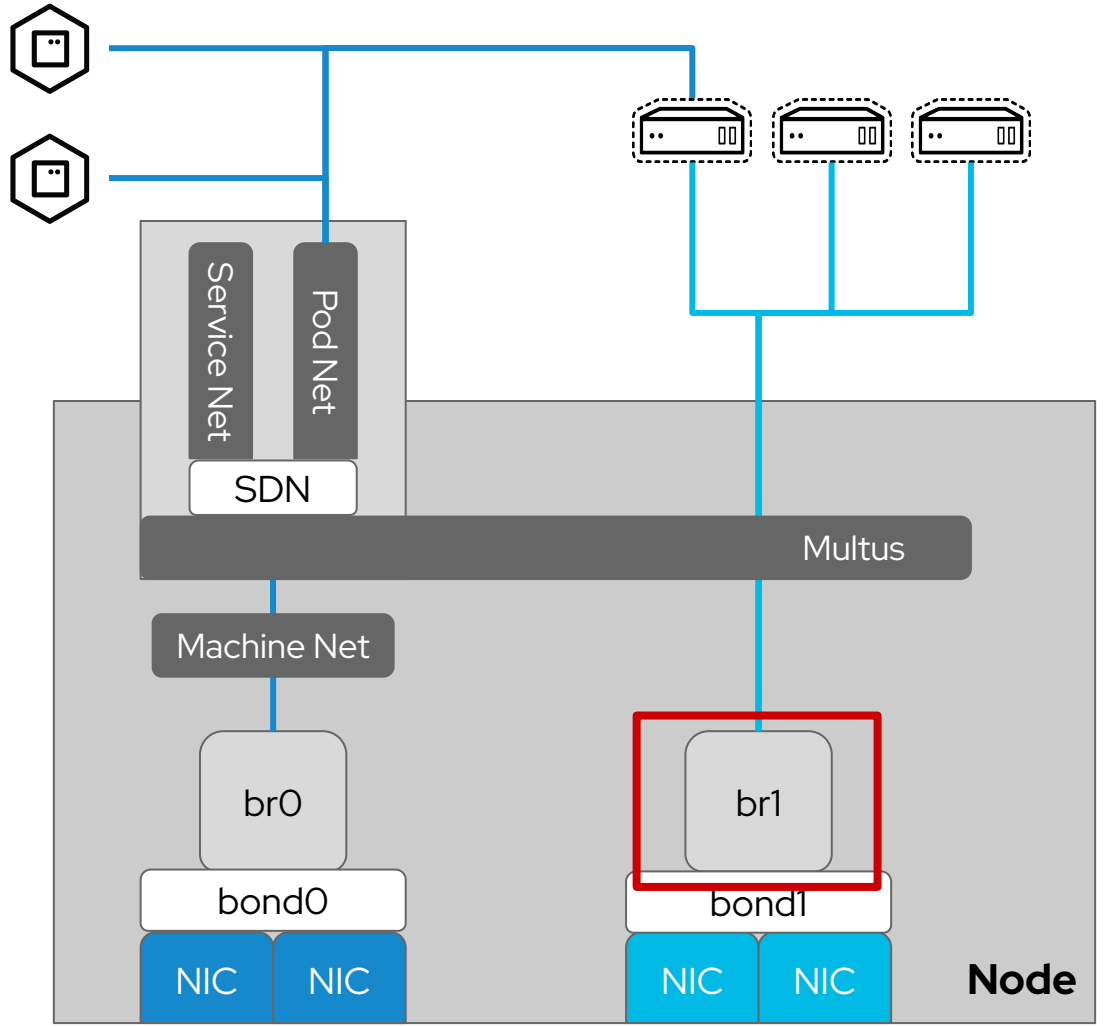
```yaml
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: worker-bond1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  desiredState:
    interfaces:
    - name: bond1
      type: bond
      state: up
      ipv4:
        enabled: false
      link-aggregation: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



Red Hat

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
```

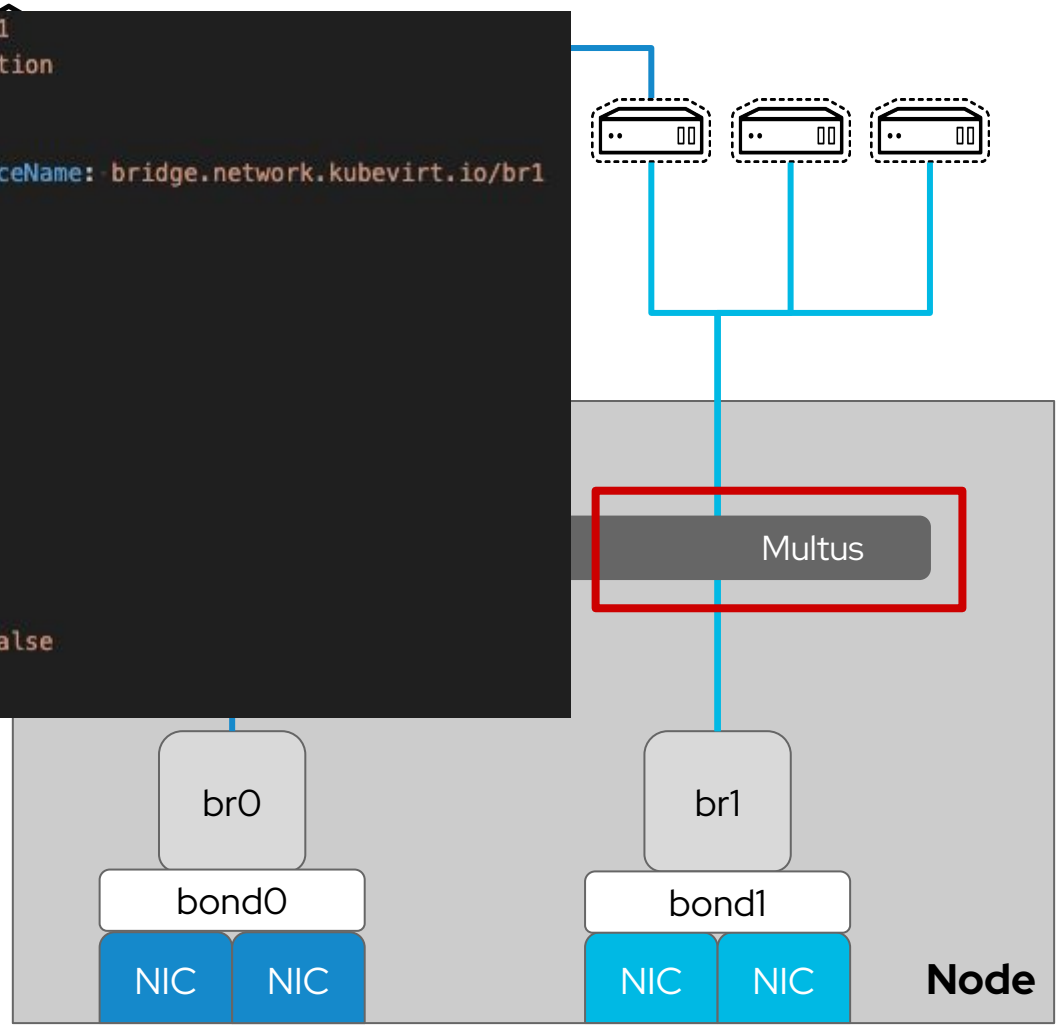


Red Hat

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

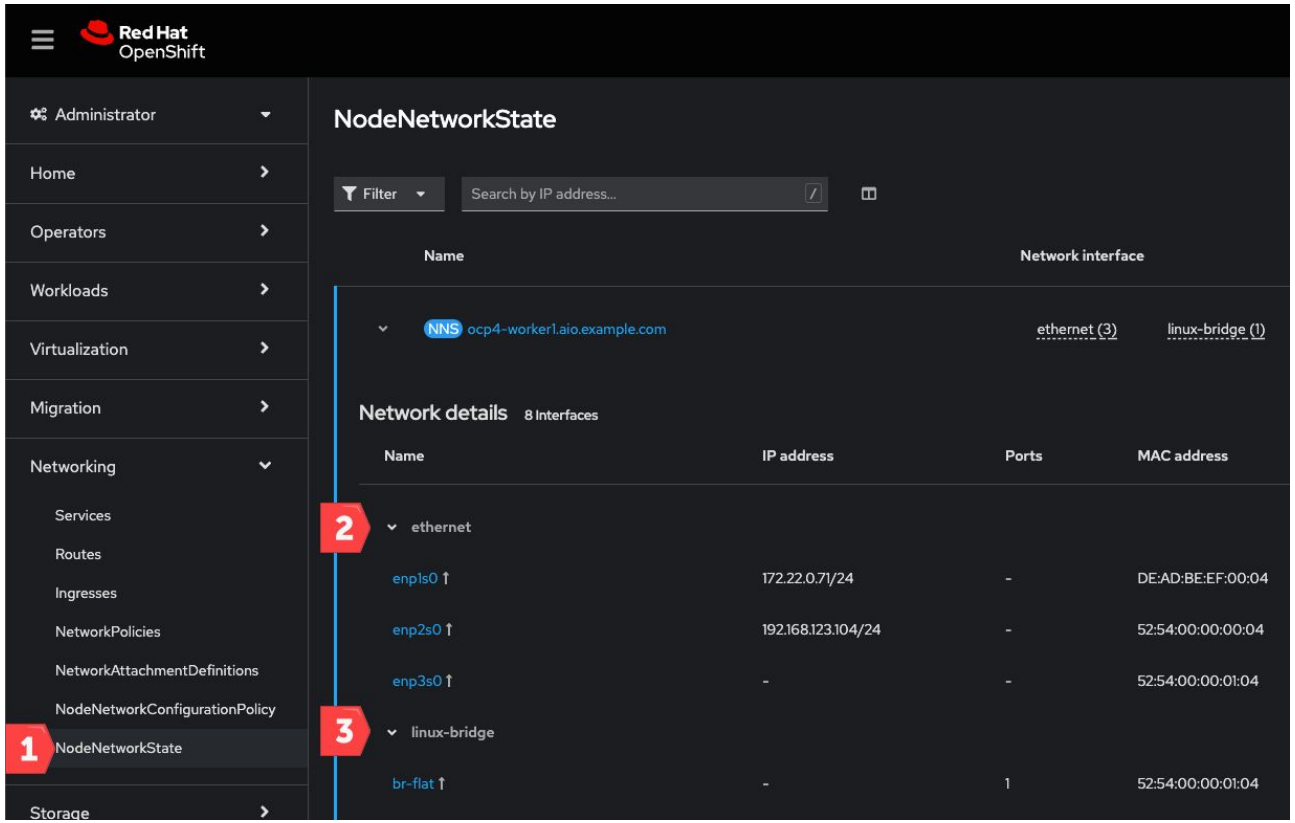
```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
  name: vlan-93
  namespace: default
spec:
  config: >-
    {
      "name":"vlan-93"
      "type":"cnv-bridge",
      "cniVersion":"0.3.1",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



Red Hat

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



Red Hat

Connecting Pods to networks

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

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

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

Connecting VMs to networks

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

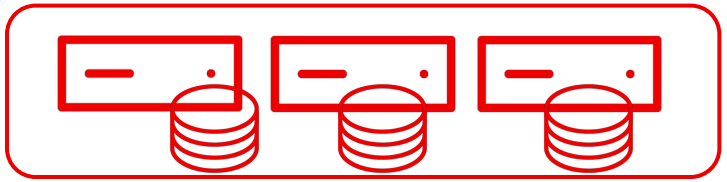
Red Hat

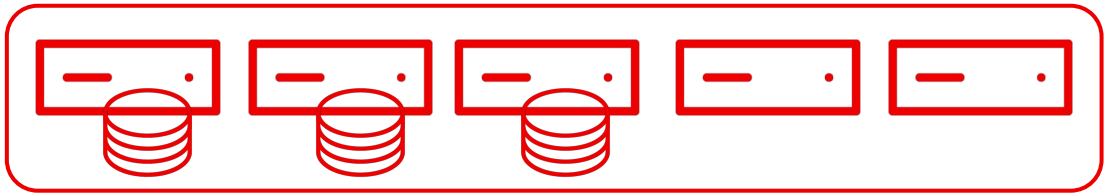
Red Hat OpenShift Data Foundation

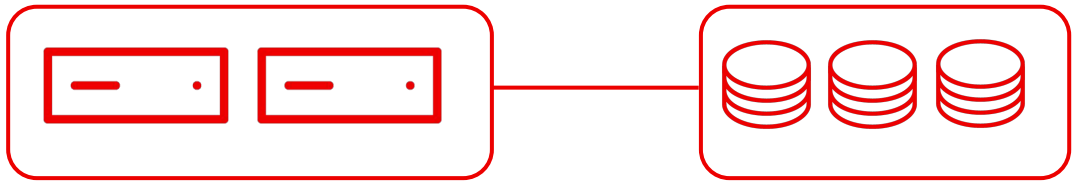
- Installed via Operator Hub
- Latest version v4.16
- Reduces complexity and increases efficiency
- Included with OPP
- Observability of PVCs health and performance
- Advanced Licensing includes Metro/Regional DR
  - Metro DR
    - ODF with external stretched Ceph Cluster
    - < 10 ms response time
    - Requires 3rd site for arbiter (< 100ms)
  - Regional DR
    - ODF Internal/HCI deployments
    - Asynchronous replication

Red Hat OpenShift Data Foundation

- Requires internal storage
- If using SAN storage:
  - Reduction in raw capacity
  - Write Amplification
  - LUN limitations
  - Storage class with replica 2
    - Reduces write amplification
    - Reduces data integrity, cannot protect against bit flip
    - Can impact performance during recovery and Increases recovery time; can only occur from single source
    - cephFS replica 2 is dev preview in v4.16
  - Multiple layers and potentially have to configure multipathing using machine configs
- Network Considerations
  - Multus for cluster and public network







Hyperconverged deployments

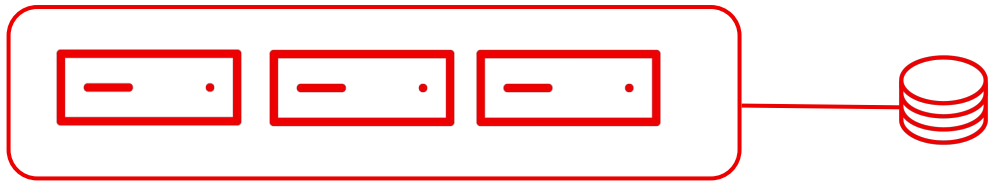
- Local storage deployed on same nodes as worker nodes
- Integrated OpenShift and storage cluster lifecycle, monitoring, and management.
- Compute and storage infrastructure scale together within the same cluster
- Optimized for simplicity of management

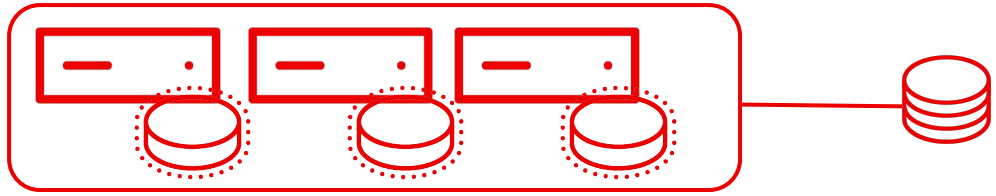
Dedicated deployments

- Local storage deployed on infrastructure or dedicated storage nodes
- Integrated OpenShift and ODF cluster lifecycle, monitoring, and management.
- Compute hosts and storage hosts scale independently within the same cluster
- Balanced

External storage

- Decoupled OpenShift and Ceph lifecycle, monitoring, and management.
- Compute and storage infrastructure scales independently in different clusters/platforms
- Optimized for scale and performance (on-premises only)





CSI Drivers

- Connect monolithic storage arrays
- iSCSI or FC
- Vendor specific features/capabilities
- Might work well for PoC or small environments
- ***Don't do for PoC or have vendor participate***

Mixed Mode

- Connect FC storage to worker nodes
- Requires CSI Driver or manual configuration of multipathing using `machineConfigs`
- Deploy ODF; select the storage class created by CSI driver
- Write amplification with ODF 3-replicas
- Reduces API calls to storage platform
- Increased performance visibility of PV/PVCs
- ***Don't do for PoC or have vendor participate***

CSI Drivers

Boot Camp 2024 - Brenda

McLaren

Red Hat

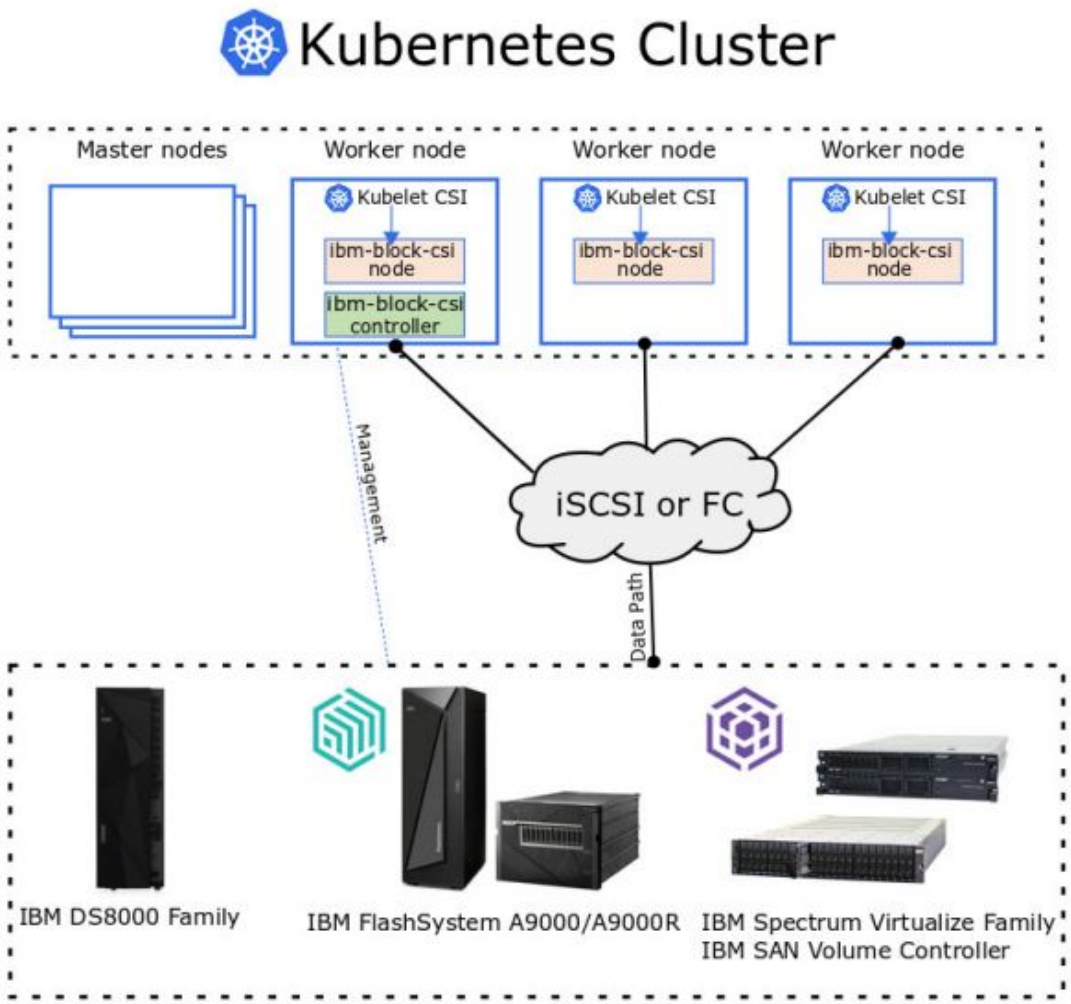
- Have the vendor participate in the PoC
- Know the limitations
  - KubeVirt Support
  - Protocols
    - iSCSI
    - NFS
    - FC
    - NVMe/TCP
  - LUN Limitations
- Know the CPU/Memory requirements
- Driver must support RWX for Live Migrations

- Requires ODF and connecting to external storage
- Latest version 1.5.0 supports OCP v4.13/4.14
- Supports Spectrum Virtualize Family
- Requires iSCSI connectivity and multipathing configured on the hosts
- Cluster capacity of .5, 2, or 4 TiB
- Limitations
  - Only x86 Architecture
  - Reports are not generated for FlashSystem information and events.
  - For direct IO to FlashSystem use the FlashSystem storage class for RWO.

Source:
https://docs.redhat.com/en/documentation/red_hat_openshift_data_foundation/4.16/html/deploying_openshift_data_foundation_in_external_mode/deploy-openshift-data-foundation-using-ibm-flashsystem#creating-an-openshift-data-foundation-cluster-for-ibm-flashsystem-storage_ibmfs-external

- Installed via Operator Hub
- Latest version 1.11.3 supports OCP v4.14/4.15
- Supports Spectrum Virtualize Family
- iSCSI, FC, NVMe/FC[(1)] Connectivity
- Default access mode is Filesystem
- Limitations:
  - Only supports RWO; RWX is enabled in v1.12 beta version
  - Contact luizgvcosta@ibm.com for external beta testing for non-production workloads.
  - Block storage only.
  - Single backend storage class

Source: https://www.ibm.com/docs/en/stg-block-csi-driver/1.11.3



- Management Path: ibm-block-csi-controller
  - Storage API Calls to backend array
- Data Path iSCSI/FC: ibm-block-csi-node
  - Mounts the volume and make available to the containers
- IBM Block CSI Driver User Guide

Red Hat

Hewlett Packard Enterprise

- Installed via Helm or Operator Hub
- Latest version 2.5.1 supports OCP v4.16
- Supports Alletra, Nimble, Primera, and 3PAR
- FC, iSCSI Connectivity
-

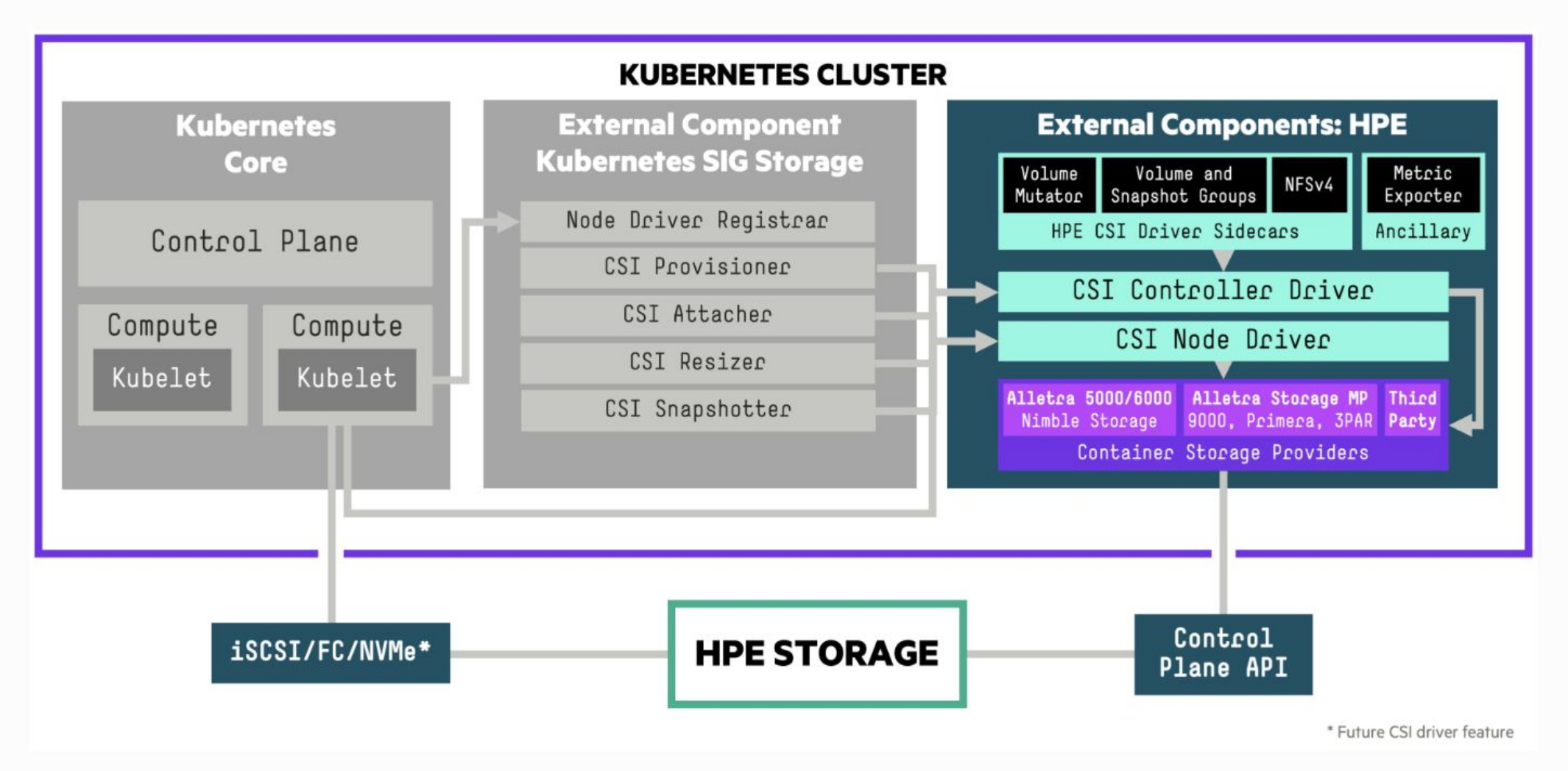
Source: https://scod.hpedev.io/partners/redhat_openshift/index.html

Red Hat

Hewlett Packard Enterprise

- Is primarily a block storage driver but includes an NFS Server Provisioner that allows RWX for `volumeMode`: `Filesystem`
- When using an FC only array and provisioning RWX block volumes, the "multi_initiator" attribute won't get set properly on the volume. The workaround is to run group --edit --iscsi_enabled yes on the Array OS CLI.
- Prerequisite - download the SCC, create namespace and apply the SCC
-

Source: https://scod.hpedev.io/partners/redhat_openshift/index.html



Source: https://scod.hpedev.io/partners/redhat_openshift/index.html

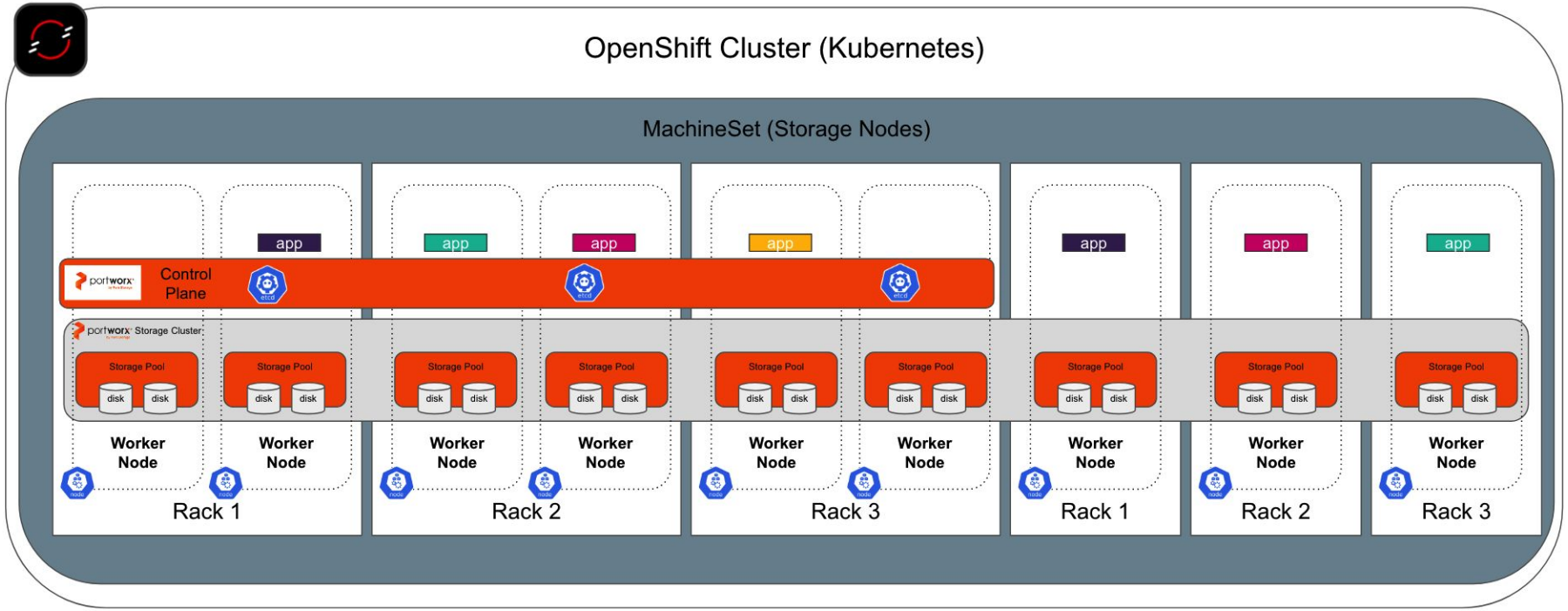
- Installed via Operator Hub
- Latest version v2.11.0 supports OCP v4.15-4.16
- Supports PowerMax, PowerStore, PowerFlex, PowerScale, Unity XT, and ObjectScale
- FC, iSCSI, NVMe/TCP Connectivity
- Additional Container Storage Modules
  - Authorization
  - Observability
  - Replication
  - Resiliency
  - Encryption Application Mobility
  - Volume Group Snapshot
- Limitations:
  - Mounts using NVMe on PowerStore fails; use v2.10.1
  -

Source: https://dell.github.io/csm-docs/docs/prerequisites/

- Editions:
  - Portworks Essentials
  - Portworks Enterprise
  - Portworks CSI for Flashblade/FlashArray (PSO replacement)(1)
- Installed via Operator Hub
- Internal/External KVDB (etcd)
  - Stores the cluster's state, configuration, data, and metadata associated with storage volumes and snapshots. Deployed on (3) nodes in the cluster
  - For highly available environments, recommended to use external etcd instance.
- Latest versions supports OCP v4.16
  - Operator 24.1.1
  - PX 3.1.3
  - STORK 24.2.4
- STORK - Storage Orchestrator for Kubernetes

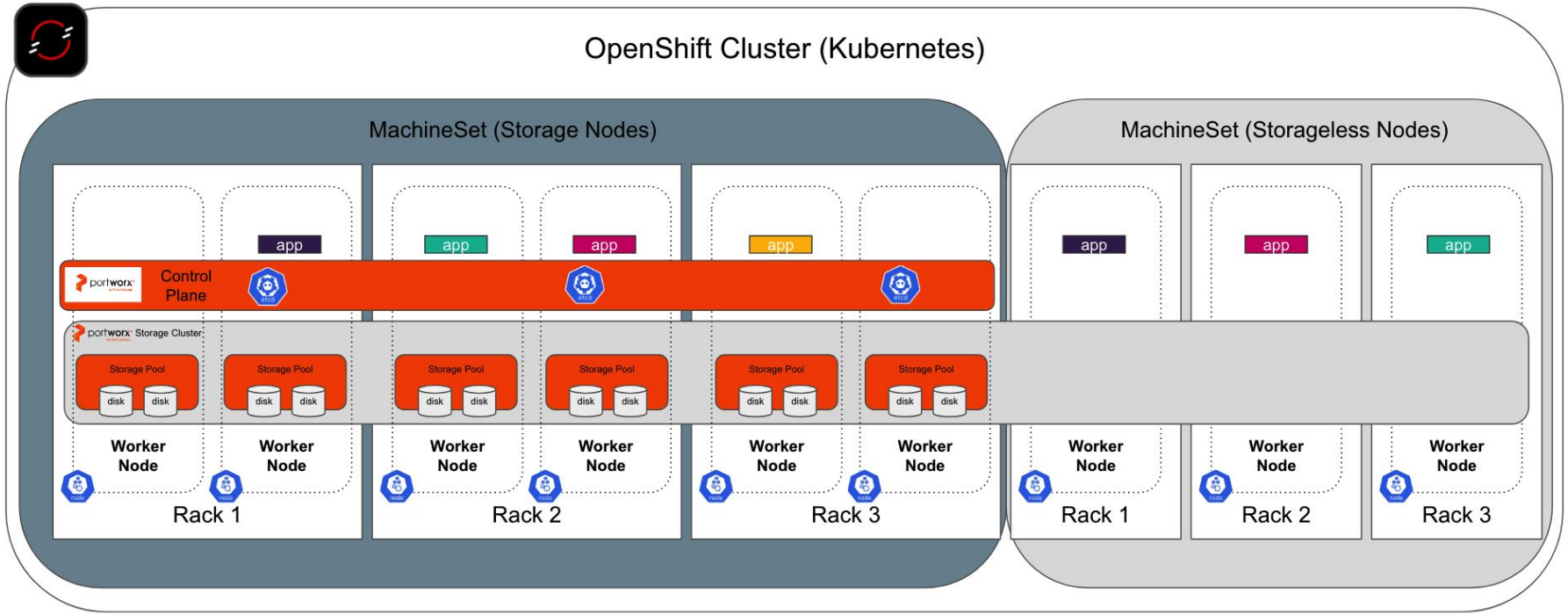
- PX CSI - does not support Kubevirt
- Mounts volumes using bind mounts or NFS v3(2)(3)
- RWX is file based
  - RWX Raw Block for Flash Array is coming Oct 2024
  - RWX block for PX Volumes is in the works (P0) and is expected to be released for Portworx Enterprise Q1/25
- RWO
  - Connected via PX control plane
  - Raw block performance
- PX Enterprise for OCPv - Directed Availability
  - Questionnaire and assessment
  - Tool that collects information from vSphere

Portworx Hyper Converged Architecture



- **Uniform Node Role:** All worker nodes serve dual roles, providing both compute and storage resources.
- **Balanced Resource Utilization:** Storage load is evenly distributed across all nodes, preventing any single set of nodes from becoming a bottleneck.
- **Integrated Scaling:** Adding more nodes to the cluster increases both compute and storage capacity simultaneously.

Red Hat

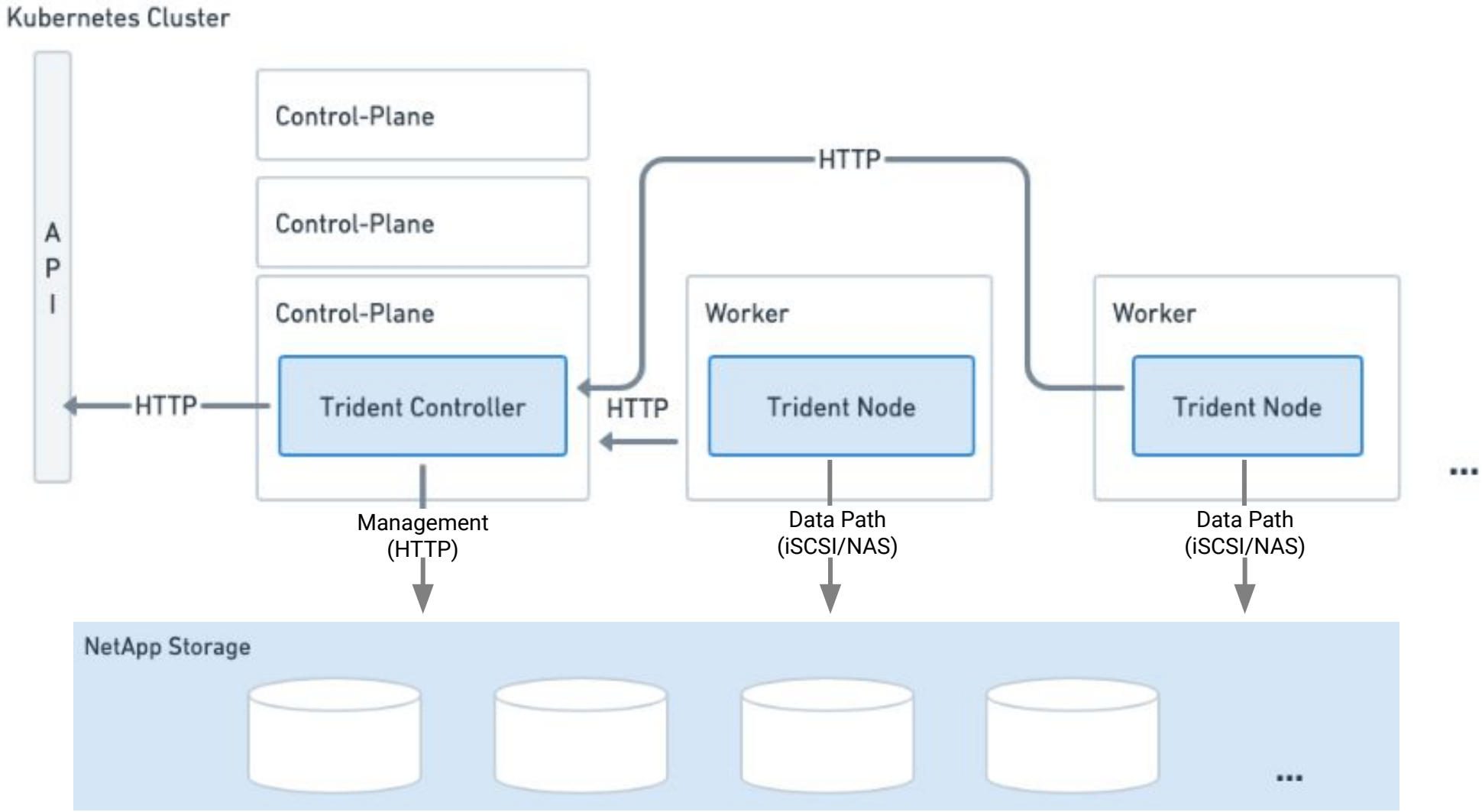


- **Dedicated Storage Nodes:** Only a selected group of nodes are responsible for storage, while the rest focus on compute tasks.
- **Resource Specialization:** Storage nodes can be optimized with hardware specifically suited for storage tasks, such as high-performance NVMe disks and additional RAM, without impacting compute nodes.
- **Isolation:** Faults or performance issues in the compute nodes do not directly affect the storage nodes, and vice versa, providing a layer of operational isolation.

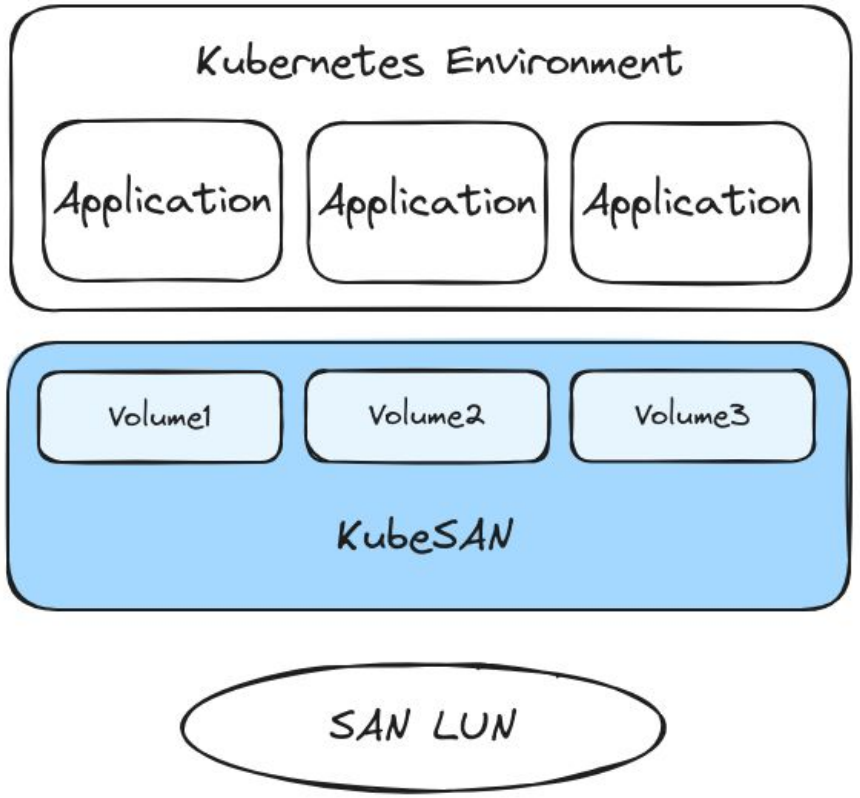
- Community based maintained by **NetApp; certified operator coming soon**
- Installation methods:
  - Manual install; download installer pkg from GitHub and use tridentctl command to install
  - Operator deployment
  - Helm
  - ArgoCD
- Runs as a single controller pod on the control plane plus a node pod (daemon set) on each worker node in the cluster
- Latest version is 24.06 supports OCP v4.16
- NFS, iSCSI, NVMe/TCP; **FC to be available Q1 2025**
- Mounts volumes using NFS v3, v4, or v4.1
- Supports KubeVirt/OCP Virtualization provisioning
  - **RWX, raw block (`volumeMode: Block`) with iSCSI or NVMe/TCP**

- Discontinuing Astra Control in lieu of Trident Protect
  - Data protection capabilities will be provided at no additional cost.
- ETA October 2024; installation using Helm or Certified Operator
- Intra-cluster and cross-cluster data protection
  - Snapshots, Backup/Restore to S3
  - On-demand and Scheduled snapshots and backups
  - Application consistency via pre/post hooks
  - DR with storage replication for lower RTO and RPO (support for VMs)
- CLI based and Kubernetes native (CR's) for easy integration with GitOps, CI/CD, IaC, and automation
- Protection of all K8S resources and PVs or just data PVs
- Self-contained backups to object storage for data residency
- Principle of least privilege security posture
- No centralized management

NetApp Trident Architecture



Red Hat



- What is it?
  - KubeSAN is a CSI plugin for Kubernetes that enables you to provision volumes backed by a single, cluster-wide, shared block device (e.g., a single big LUN on a SAN), exposed as a shared Volume Group under lvm2.
- Why use it?
  - No vendor CSI plugin is available, the vendor plugin is broken, or lacks features.
  - The vendor CSI plugin cannot be used due to organizational reasons (storage team does not want to give Kubernetes team access).
  - 1 SAN LUN per PersistentVolume is undesirable for scalability or organizational reasons.
- Shared LVM Volume Group and requires lvm2-lockd and sanlock
- Features:
  - Dynamic provisioning of block (RWX) and file volumes
  - Snapshots/Cloning
  - Thin Provisioning
- Roadmap
  - Recovery after power failure. Currently requires manual intervention.
  - Volume expansion
  - Instant volume cloning via background copy

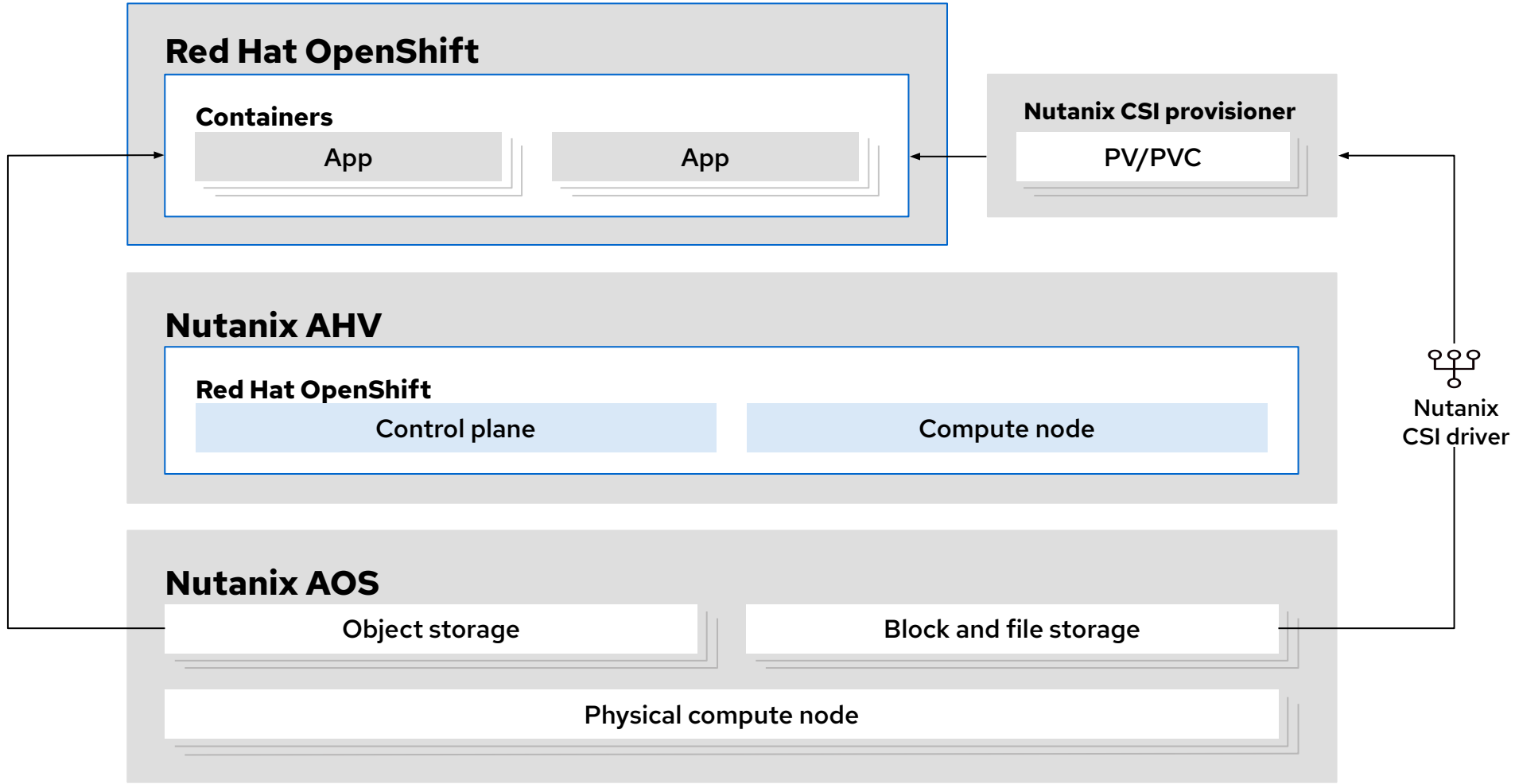
Comparing with traditional virtualization platforms

Red Hat

NUTANIX

- Strong partner we've worked with on OpenShift deals
- Many joint sales motions in the past couple of years
- OpenShift was a preferred platform for containerization on Nutanix - they are now positioning their own k8s platform
- Technical sales teams are good, motivated and positive
- Very limited solution in terms hardware
- Closer to VMWare in day-to-day operations

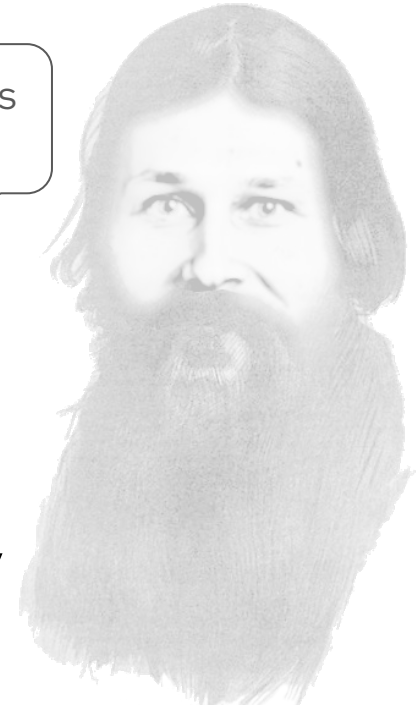
NUTANIX



Red Hat

vmware	Platform Capabilities	NUTANIX
VMC	Hybrid Cloud	Nutanix Clusters (NC2)
Horizon	End User Computing / Virtual Desktops	Nutanix Desktop Services
	Database Services	Nutanix Databases (NDB)
Aria Migration / HCX	App Mobility / Migration	Move
Aria Cost by CloudHealth	Cost / Governance	Nutanix Cloud Management (NCM)
Aria Automation (vRA)	Application Automation / Orchestration / Self Service	
Aria Operations (vROPS)	Advanced Intelligent Operations	
Aria Insights	Data Analytics	Nutanix Unified Storage (NUS)
vSAN File Services	Storage Services	
NSX	Microsegmentation / App Firewalling	Nutanix Cloud Infrastructure (NCI)
	Network Overlay	
Site Recovery Manager	Advanced Replication and DR	
Tanzu Kubernetes Grid	Kubernetes	
vSAN	Storage Operating System	
ESXi	Hypervisor	

openstack™



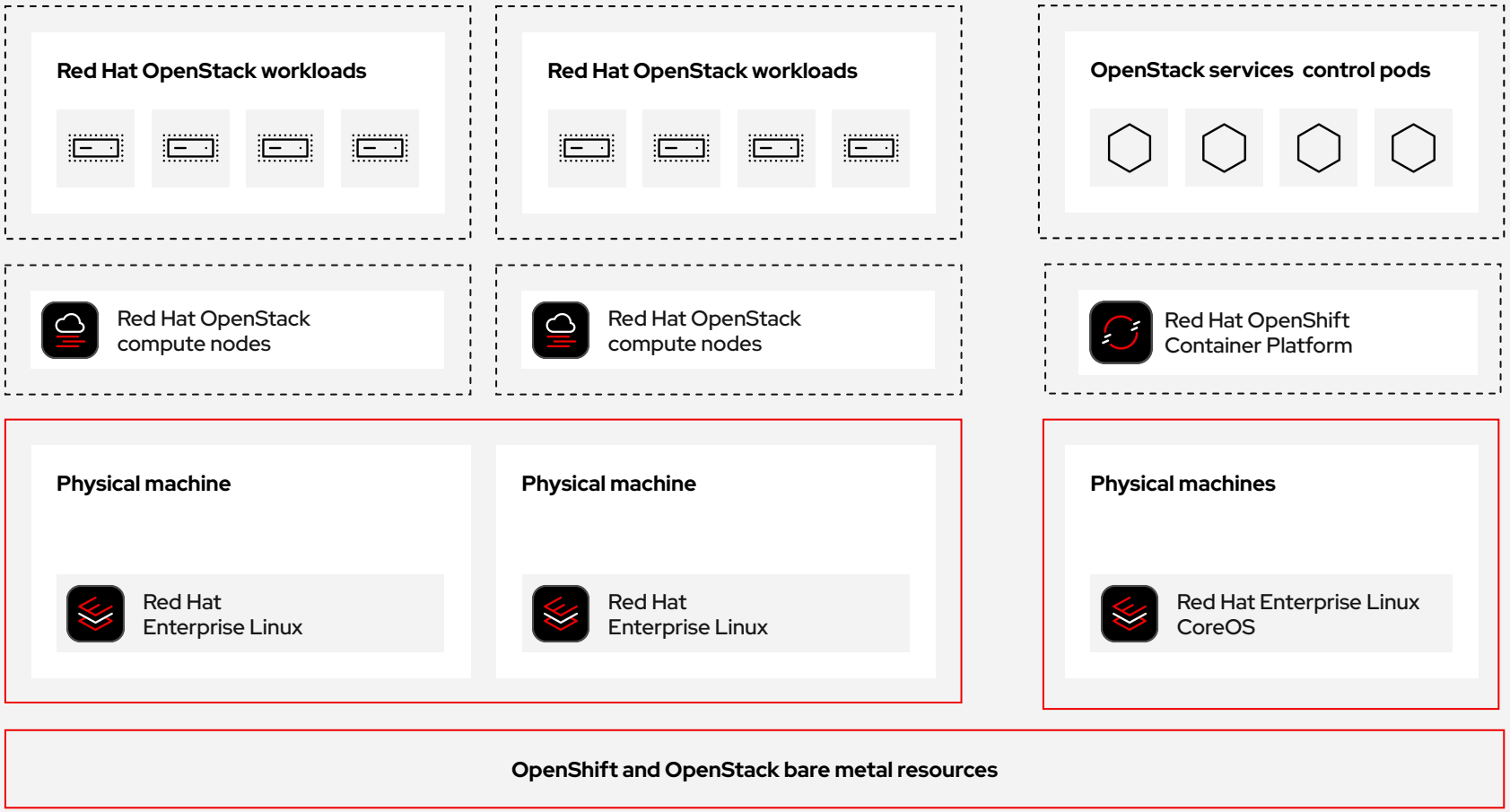
- Contrary to rumors of its demise, OpenStack is not dead
- Canonical & Mirantis are both in deals we've seen recently
- Red Hat OpenStack is still the King
- Telco workflows are still heavily dependent on OpenStack, worldwide
- While it isn't moving as fast, OpenStack *will* continue to grow
- In some cases, OpenStack can be a viable option, or a distraction
- You have to manage this, be realistic and transparent about the challenges
- OpenStack 18 brings new opportunities & challenges

Meet Red Hat OpenStack Services on OpenShift

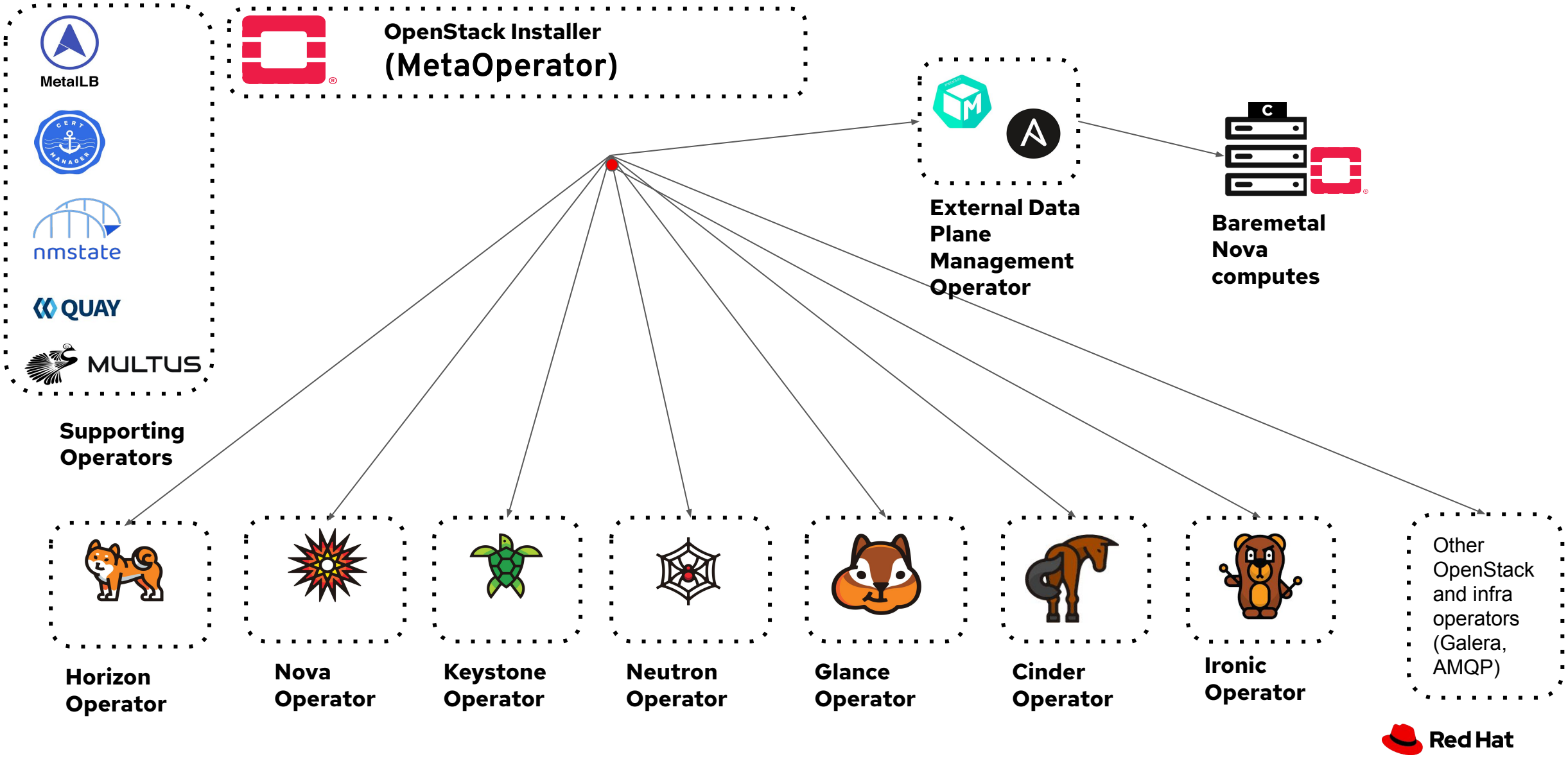
Red Hat

Red Hat OpenStack Services on OpenShift

High level diagram



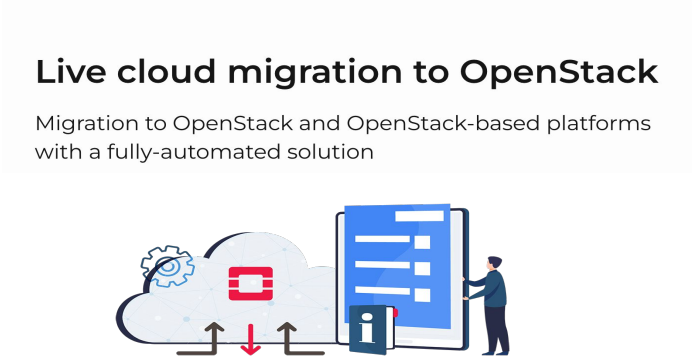
Openstack Services Deployment : High Level View



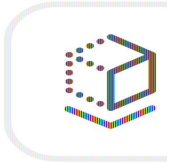
Red Hat OpenStack offers a compelling platform alternative to run VMs, specifically for workloads that require IaaS, large scale, multi-tenancy, and host isolation, suitable for large service providers such as Telco/NFV use cases, or on-prem sovereign clouds.

Migration Partners

Offering a seamless software-based solutions to migrate live production clouds without disrupting ongoing operations via certified third-party Migration Tools, such as Trilio, Cloudbase and Hystax.



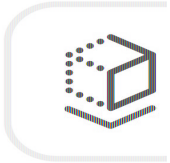
Red Hat



vmware® by Broadcom

- While aggressively taking advantage, they are still in accounts, making deals
- Customers will sometimes be using us as leverage against price hikes
- It’s up to us to differentiate ourselves, while being accessible in comparison
- A Customer’s move is often price motivated, sometimes political, but rarely technical
- This will go on for years…

What we see in the field...



vmware® by Broadcom

- **“It’s not my decision”**

  The business owners are usually to drivers of this motion, leaving the operations teams to come up with a solution. When you work with them, empathize and help them to understand our platform, its strengths and roadmap

- **Side-by-Side Feature comparison**

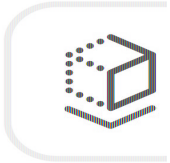
  This is the most common scenario, where it takes your willingness to be creative, transparent and informed on the similarities and differences between the two platforms

- **Sunk Costs**

  Many migrations are expected to happen on their current “sunk cost” hardware. They’re likely being hit by a large bill from Broadcom, so buying all new hardware isn’t something they want to even consider. Use our ecosystem partners to drive our ability to accommodate these customers and find solutions that will alleviate their pain

Red Hat

Keys to Success



vmware® by Broadcom

- **Know the VMware Platform**

  You must be able to speak their language if you truly want to understand their business and operational needs

Map vSphere features to OpenShift Virtualization

VMWare Learning

- **The New Way**

  OpenShift Virtualization is not a clone of VMWare, but rather, a new way of deploying, managing and integrating virtualization into a modern **gitops workflow**, powered by Kubernetes and KVM and delivered on OpenShift.

Managing Virtual Machines with Red Hat OpenShift Virtualization (DO316) v4.14

Build it and they will come

Red Hat

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

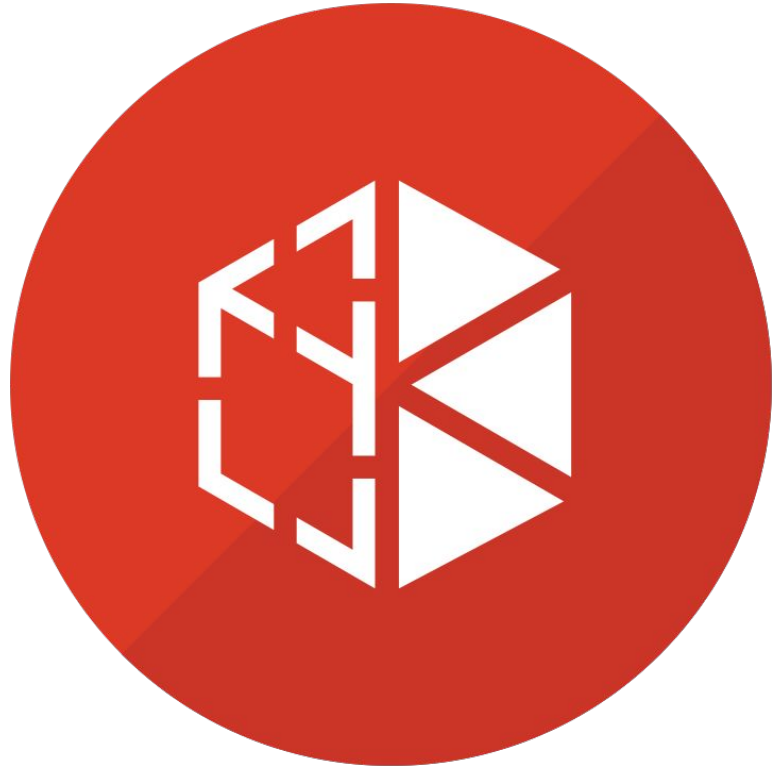
Feature	RHV	OpenShift Virtualization	vSphere
Where VM disks are stored	Storage Domain	PVC	datastore
Policy based storage	None	StorageClass	SPBM
Non-disruptive VM migration	Live migration	Live migration	vMotion
Non-disruptive VM storage migration	Storage live migration	N/A	Storage vMotion
Active resource 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

Red Hat

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

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