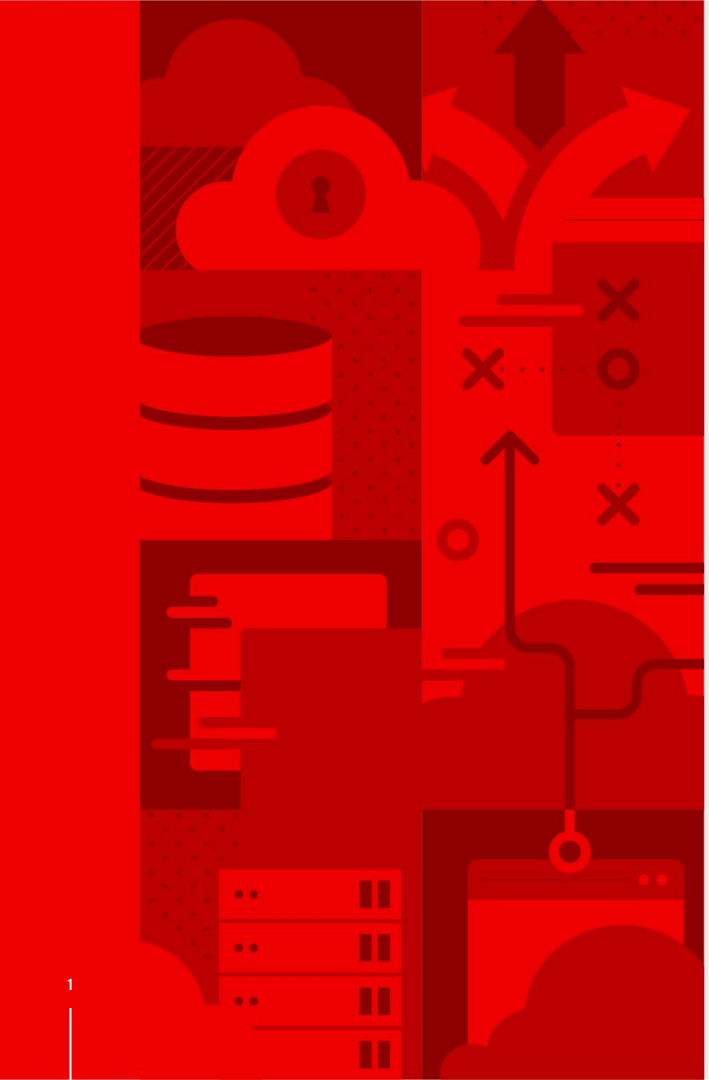


OPENSHIFT CONTAINER PLATFORM

TECHNICAL OVERVIEW

linkedin.com/company/red-hat

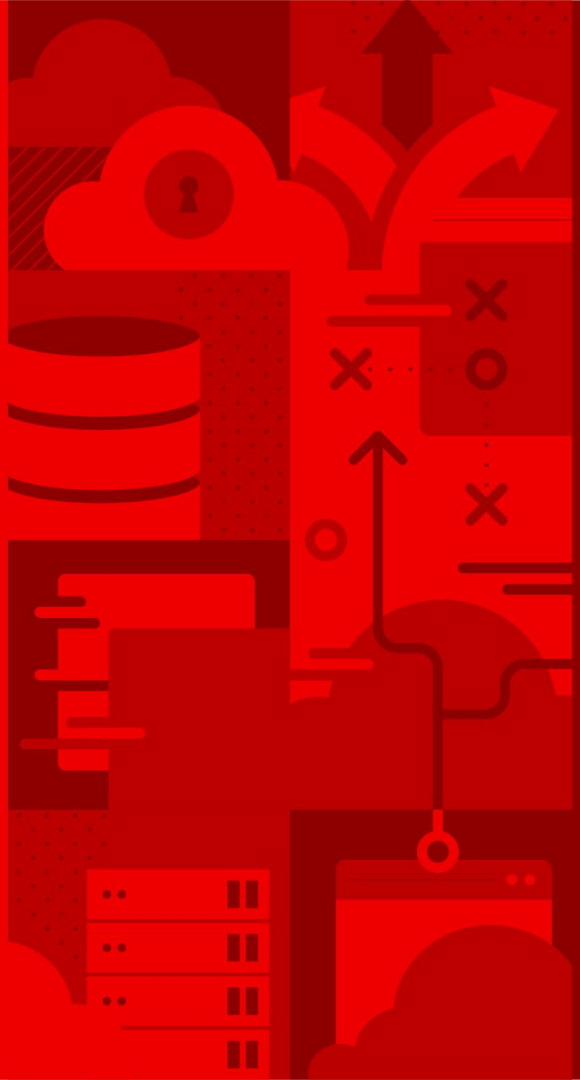
facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

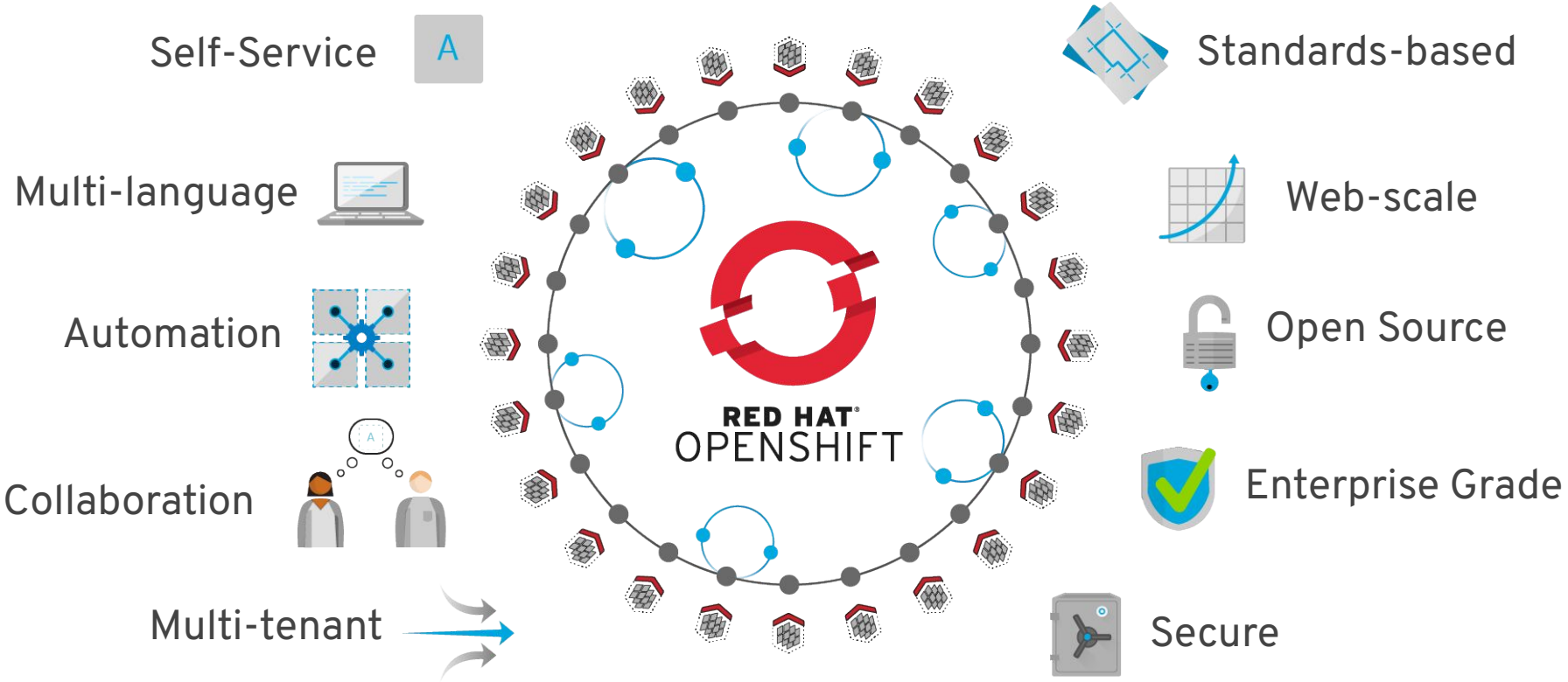
Alfred Bach
Principal Solution Architect
November 2022

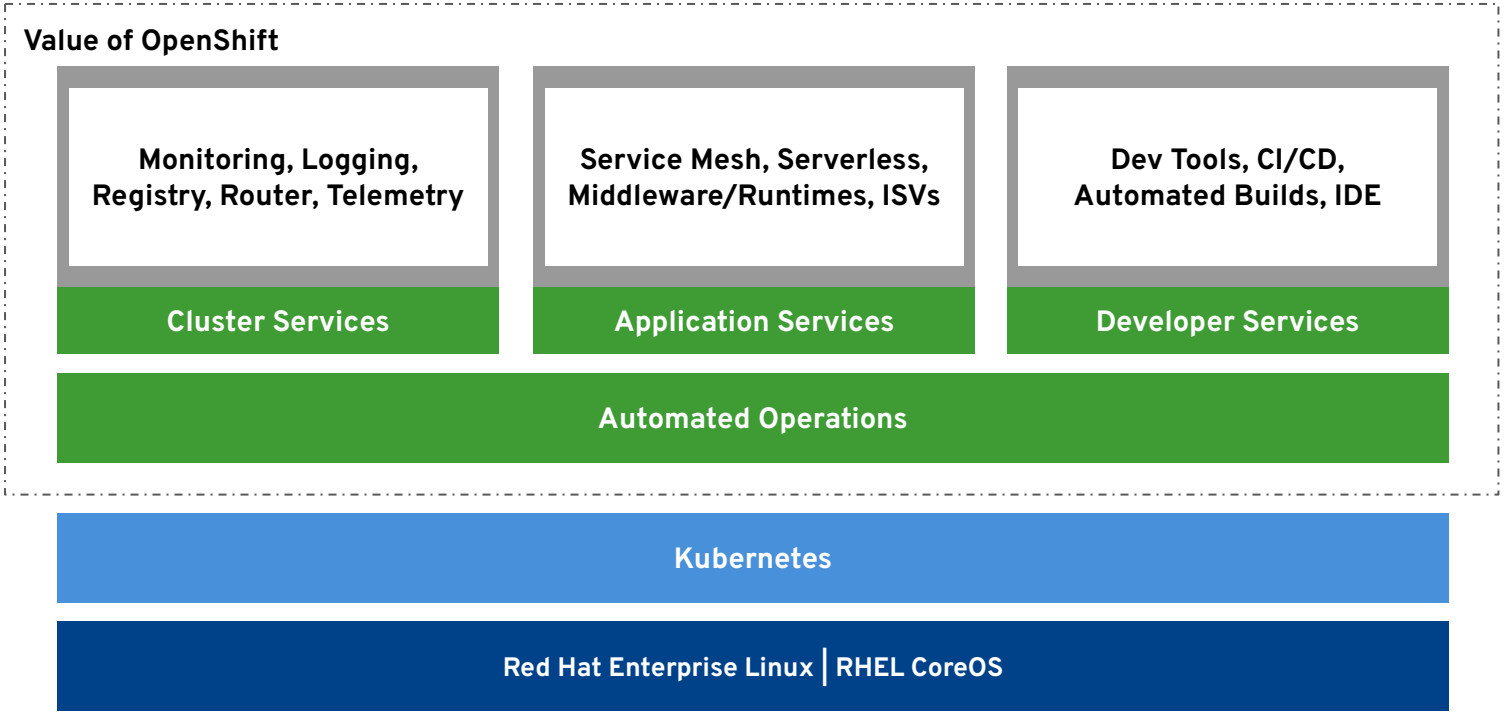
Red Hat



Functional overview

Red Hat


Self-Service
Multi-language
Automation
Collaboration
Multi-tenant
RED HAT
OPENSHIFT
Standards-based
Web-scale
Open Source
Enterprise Grade
Secure

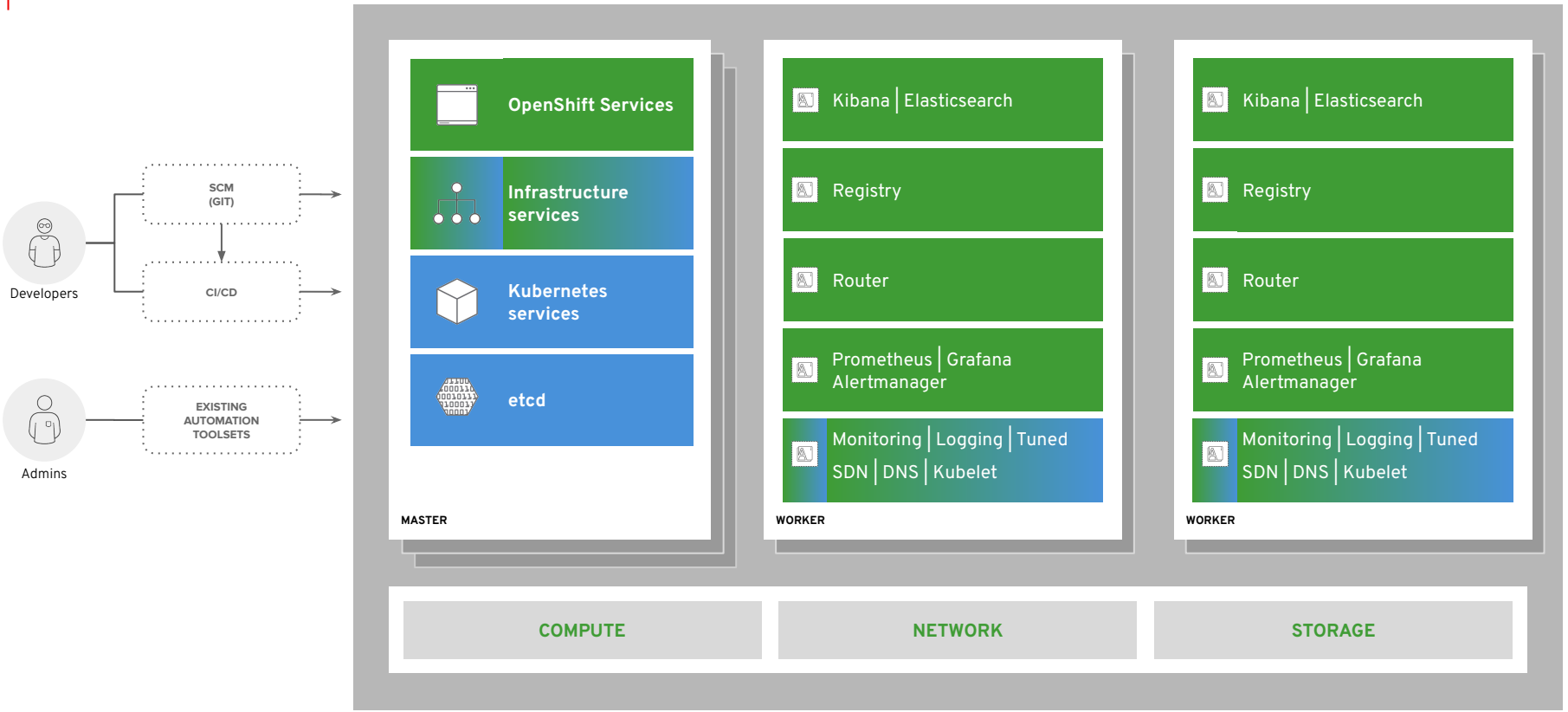


Best IT Ops Experience

CaaS ⟷ PaaS ⟷ FaaS

Best Developer Experience

Red Hat


SCM (GIT)
CI/CD
Developers
EXISTING AUTOMATION TOOLSETS
Admins
OpenShift Services
Infrastructure services
Kubernetes services
etcd
MASTER
Kibana | Elasticsearch
Registry
Router
Prometheus | Grafana Alertmanager
Monitoring | Logging | Tuned
SDN | DNS | Kubelet
WORKER
Kibana | Elasticsearch
Registry
Router
Prometheus | Grafana Alertmanager
Monitoring | Logging | Tuned
SDN | DNS | Kubelet
WORKER
COMPUTE
NETWORK
STORAGE

Red Hat

CNCF Cloud Native Landscape
2019-07-12T03:51:02Z 024e93b

Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io

Greyed logos are not open source

App Definition and Development

Database

Streaming & Messaging

Application Definition & Image Build

Continuous Integration & Delivery

Orchestration & Management

Scheduling & Orchestration

Coordination & Service Discovery

Remote Procedure Call

Service Proxy

API Gateway

Service Mesh

Runtime

Cloud-Native Storage

Container Runtime

Cloud-Native Network

Provisioning

Automation & Configuration

Container Registry

Security & Compliance

Key Management

Cloud

Public

Special

Kubernetes Certified Service Provider

Platform

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

PaaS/Container Service

Kubernetes Training Partner

Observability and Analysis

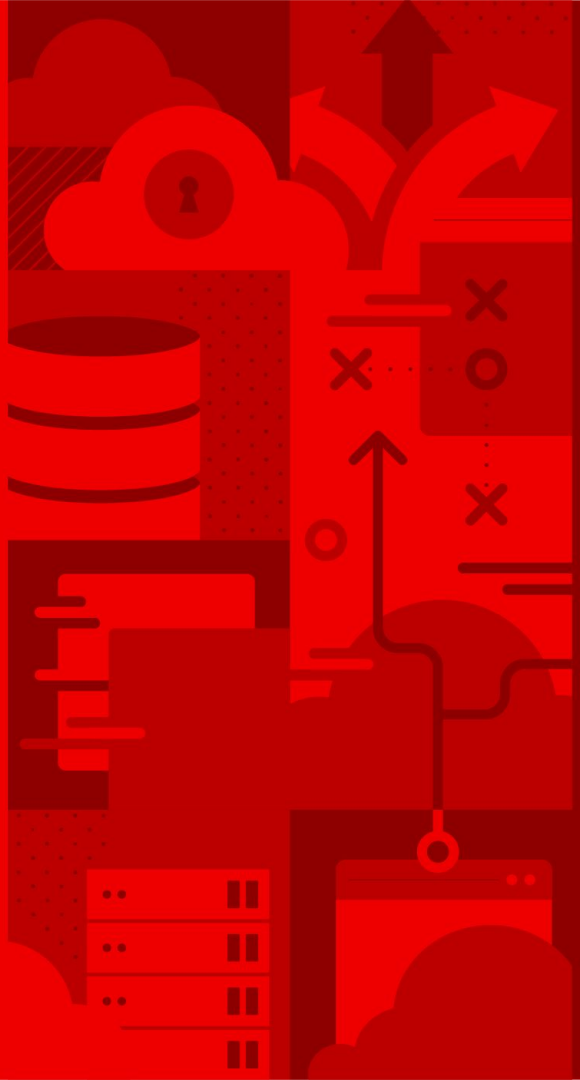
Monitoring

Logging

Tracing

Chaos Engineering

Serverless

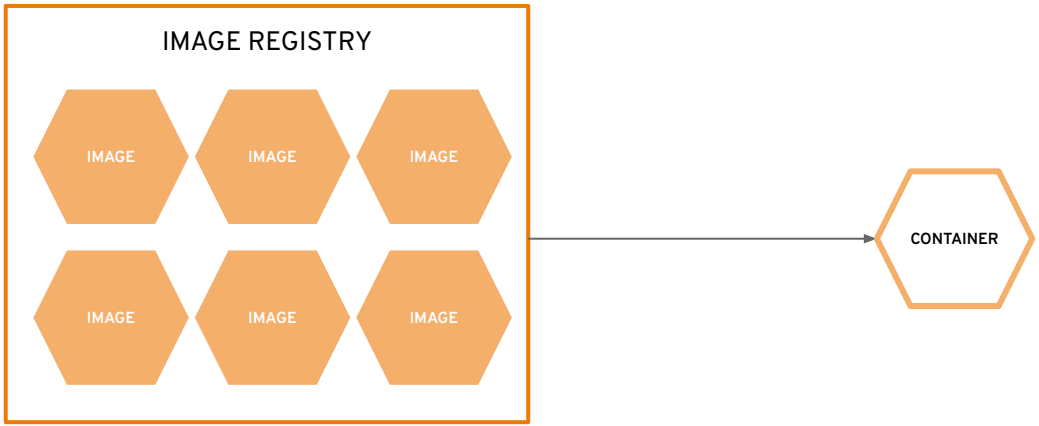


OpenShift and Kubernetes core concepts

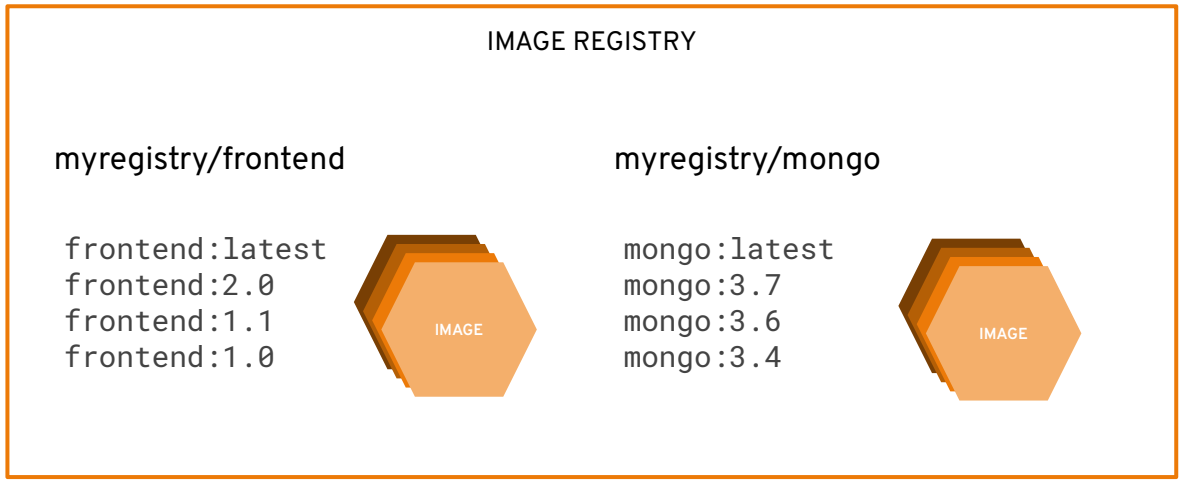
a container is the smallest compute unit

containers are created from container images

container images are stored in an image registry

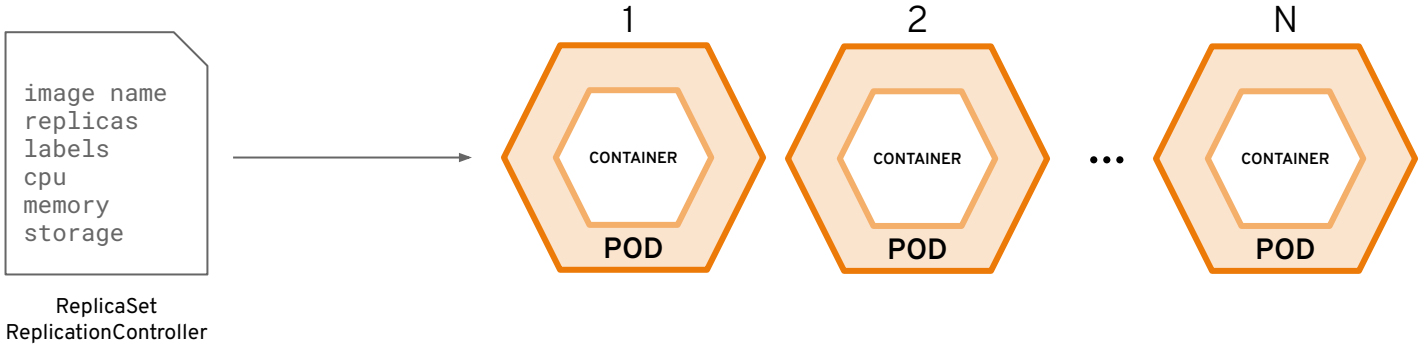


an image repository contains all versions of an image in the image registry

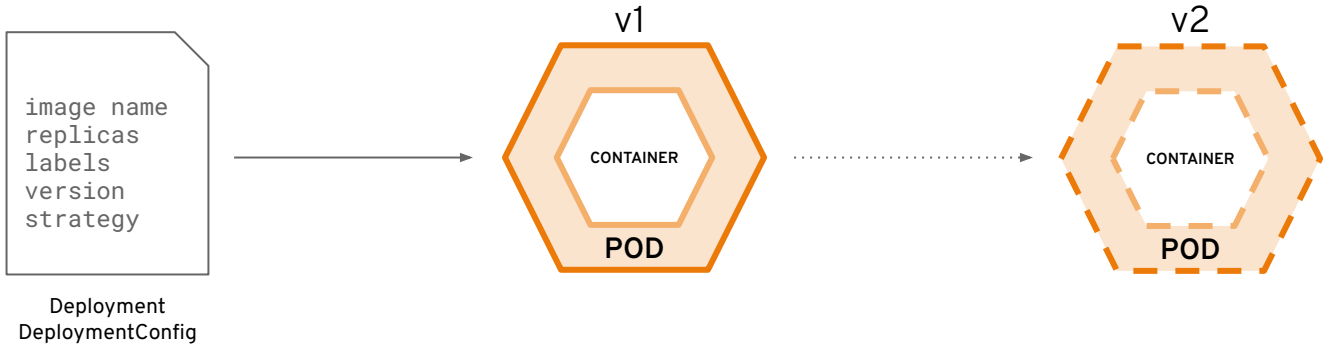


containers are wrapped in pods which are units of deployment and management

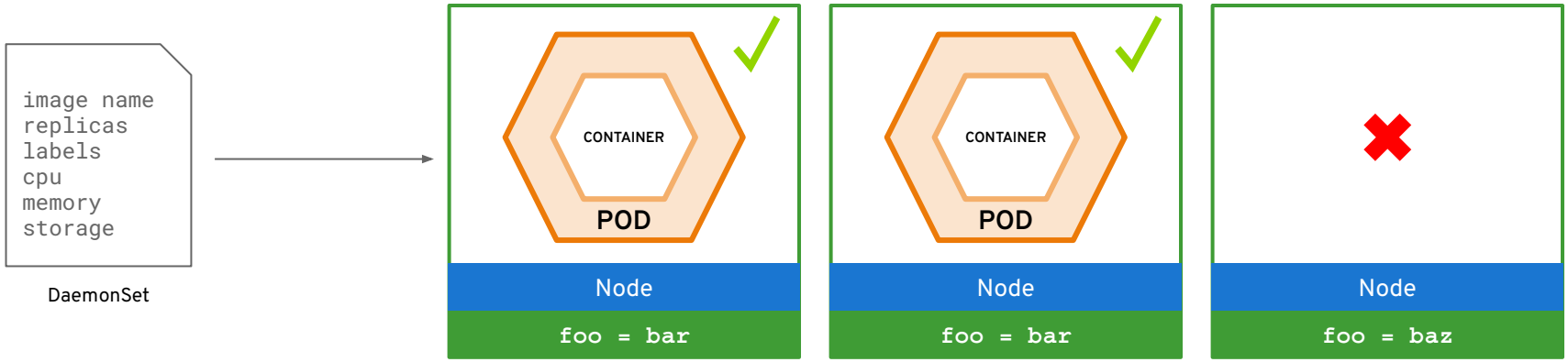
`ReplicationControllers` & `ReplicaSets` ensure a specified number of pods are running at any given time



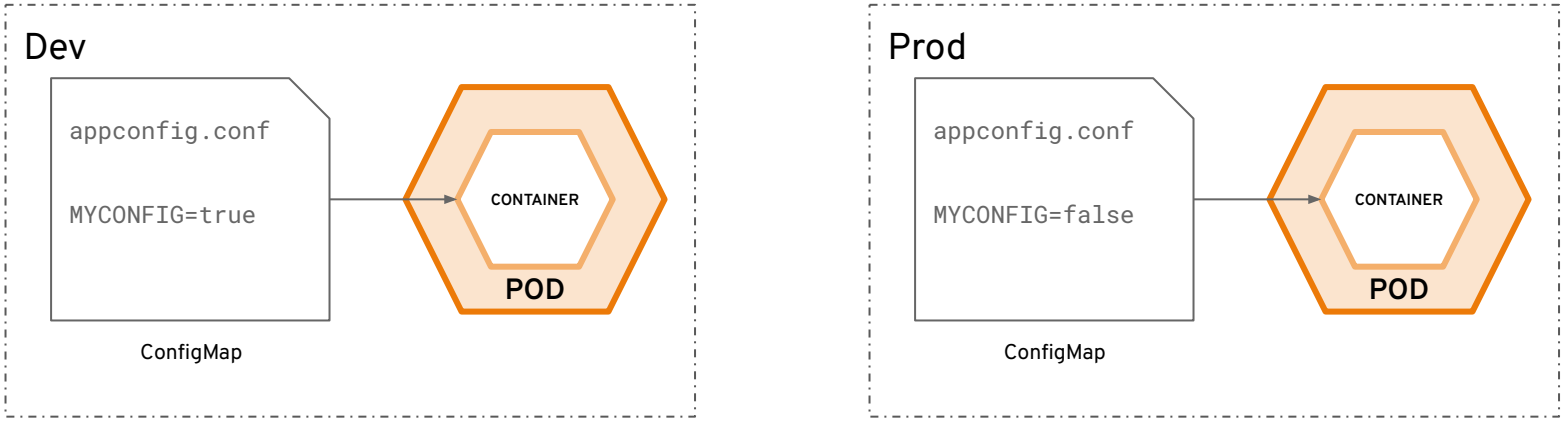
`Deployments` and `DeploymentConfigurations` define how to roll out new versions of Pods



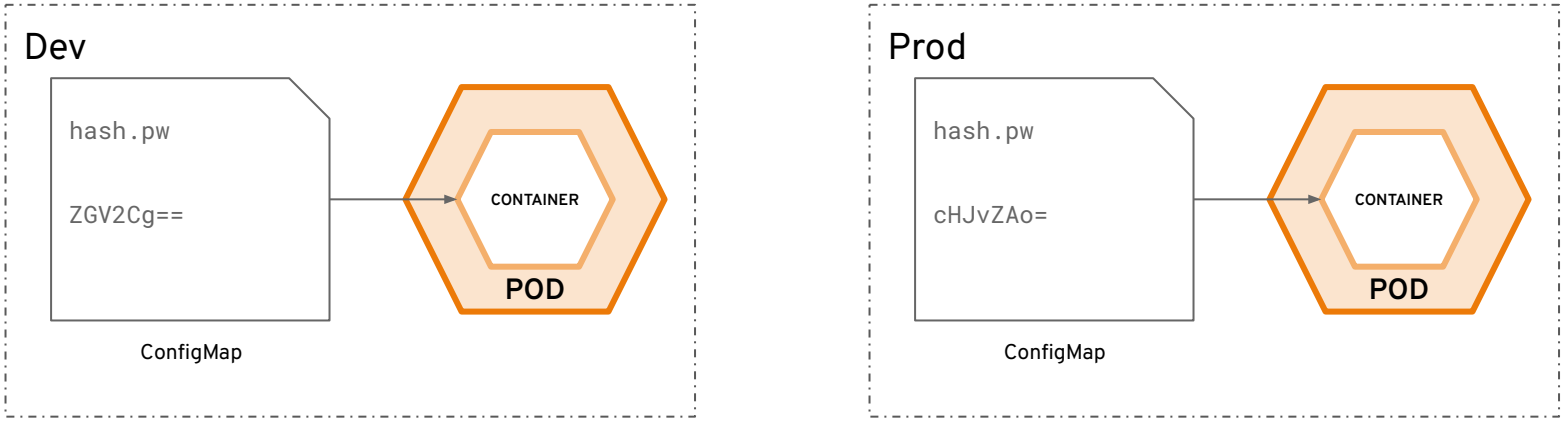
a daemonset ensures that all (or some) nodes run a copy of a pod



`configmaps` allow you to decouple configuration artifacts from image content

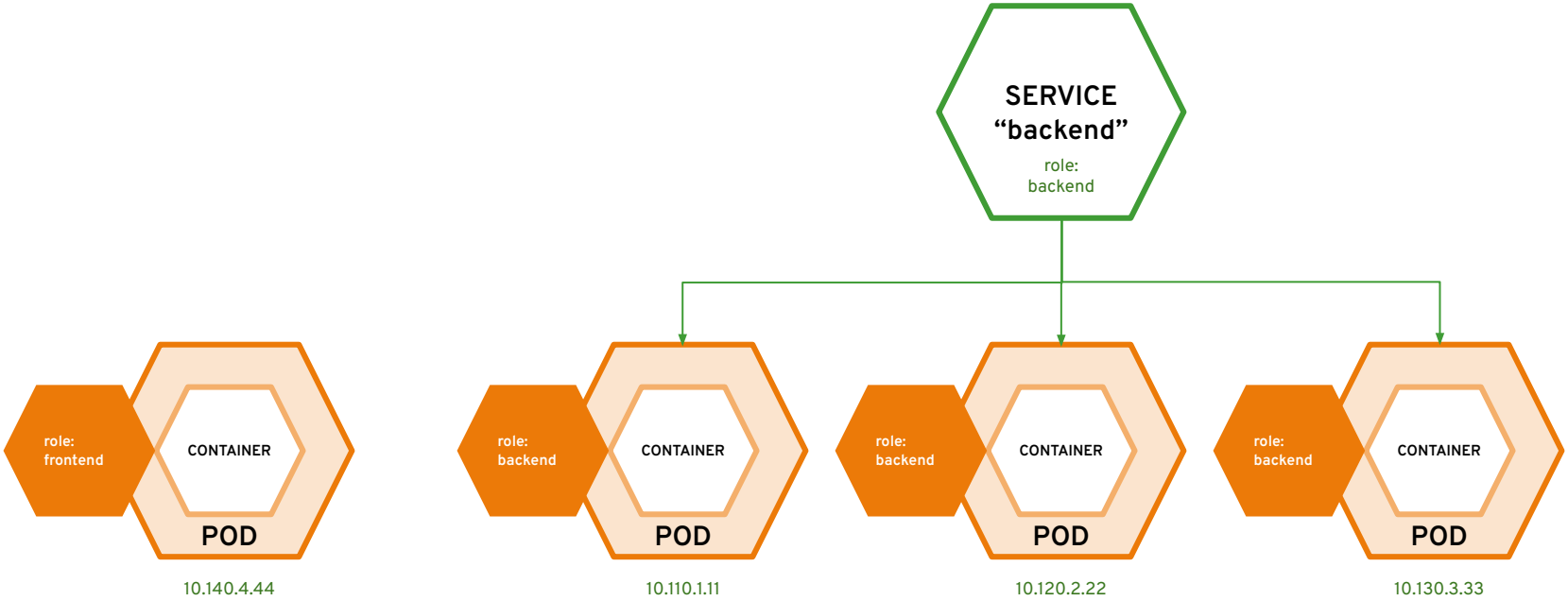


`secrets` provide a mechanism to hold sensitive information such as passwords

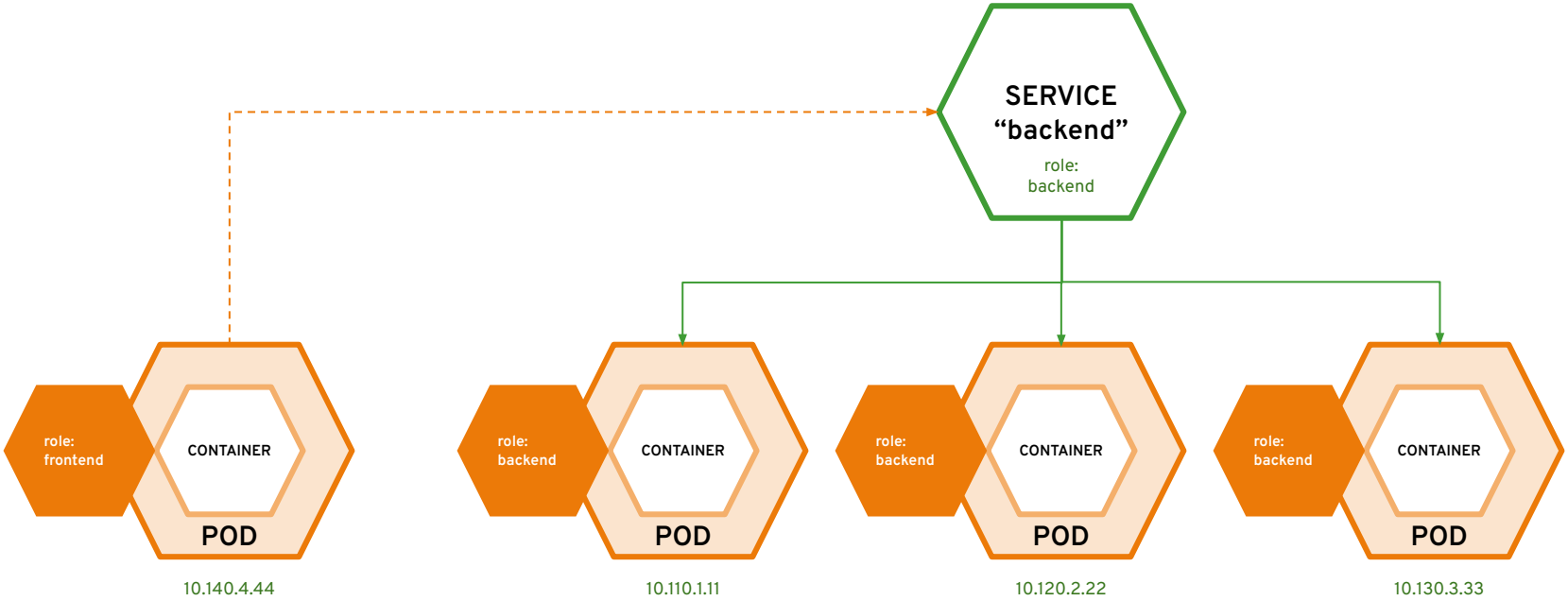


The etcd datastore can be encrypted for additional security
https://docs.openshift.com/container-platform/4.6/security/encrypting-etcd.html

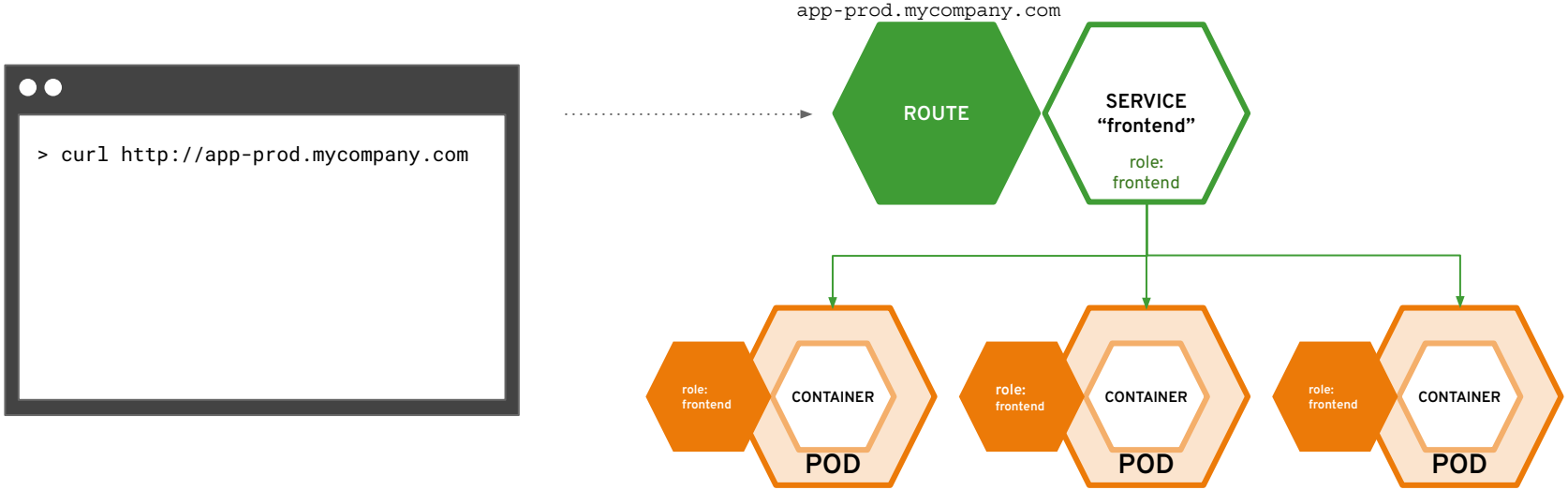
services provide internal load-balancing and service discovery across pods



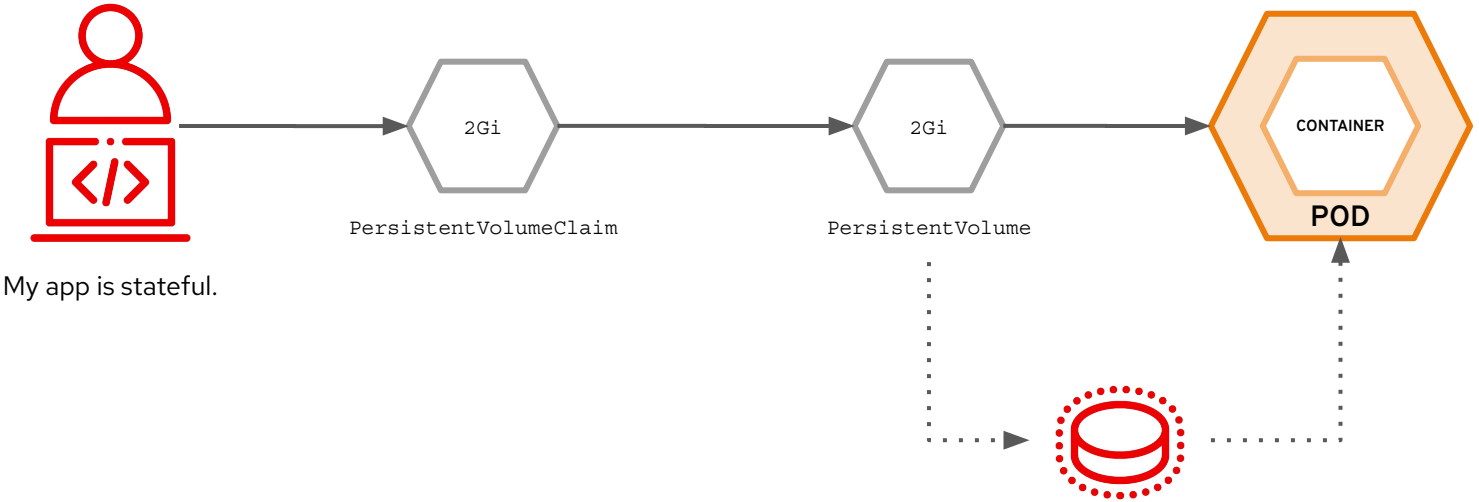
apps can talk to each other via services



`routes` make services accessible to clients outside the environment via real-world urls

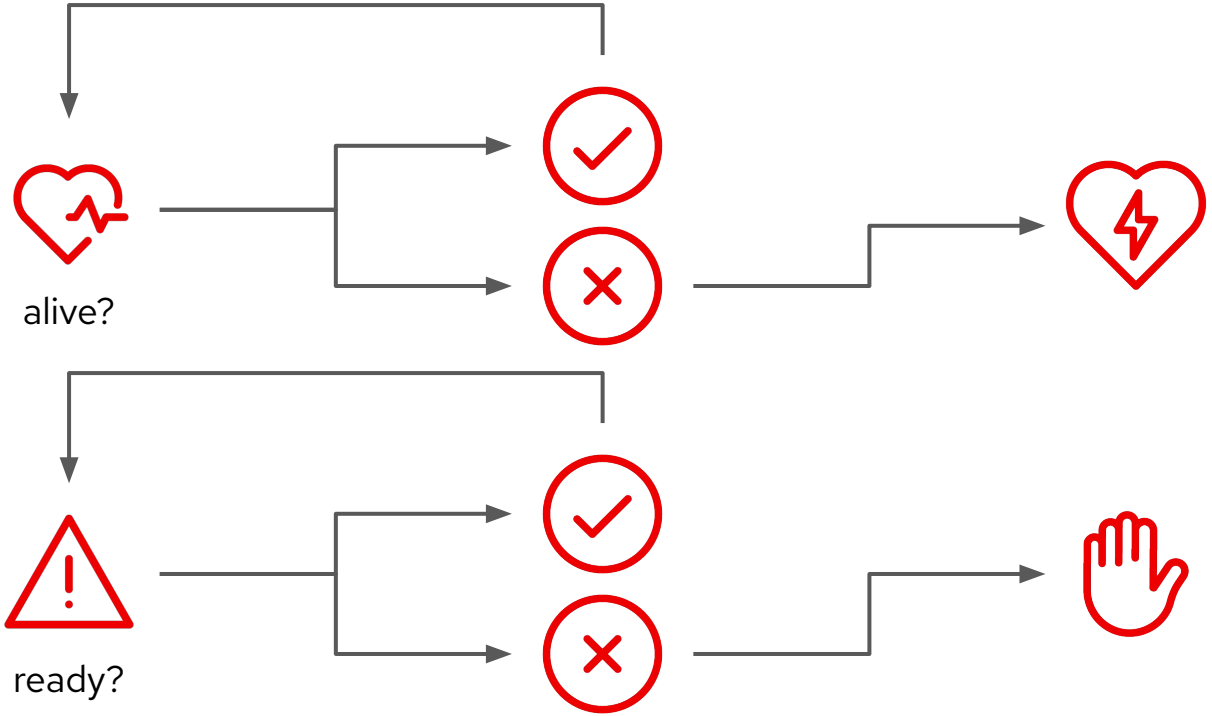


Persistent Volume and Claims

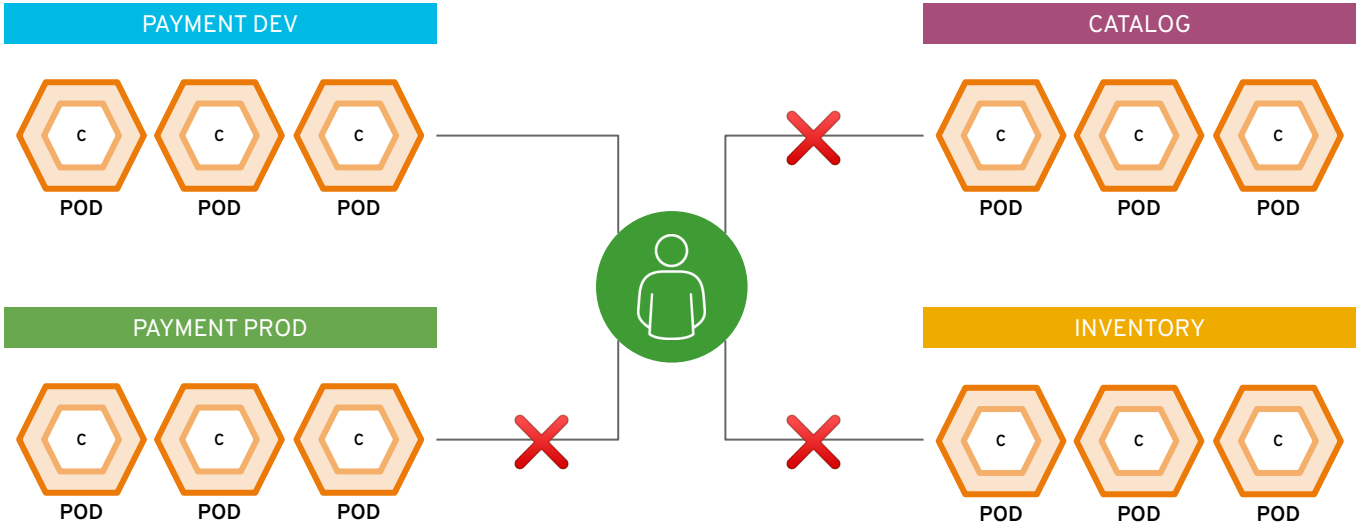


My app is stateful.

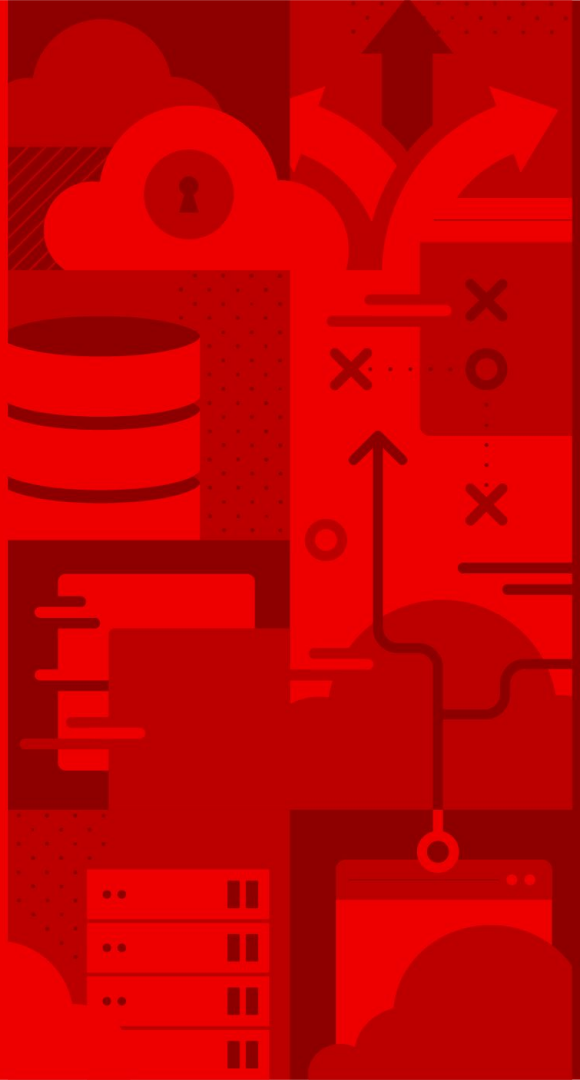
Liveness and Readiness



projects isolate apps across environments, teams, groups and departments



Red Hat



OpenShift 4 Architecture

your choice of infrastructure

COMPUTE

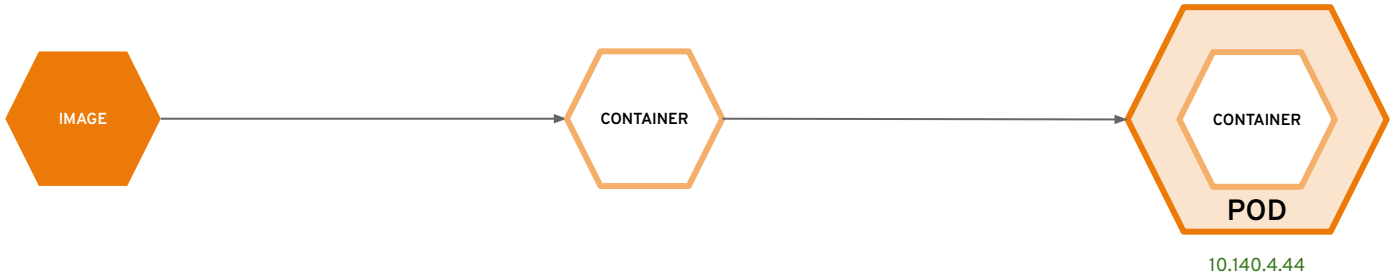
NETWORK

STORAGE

workers run workloads

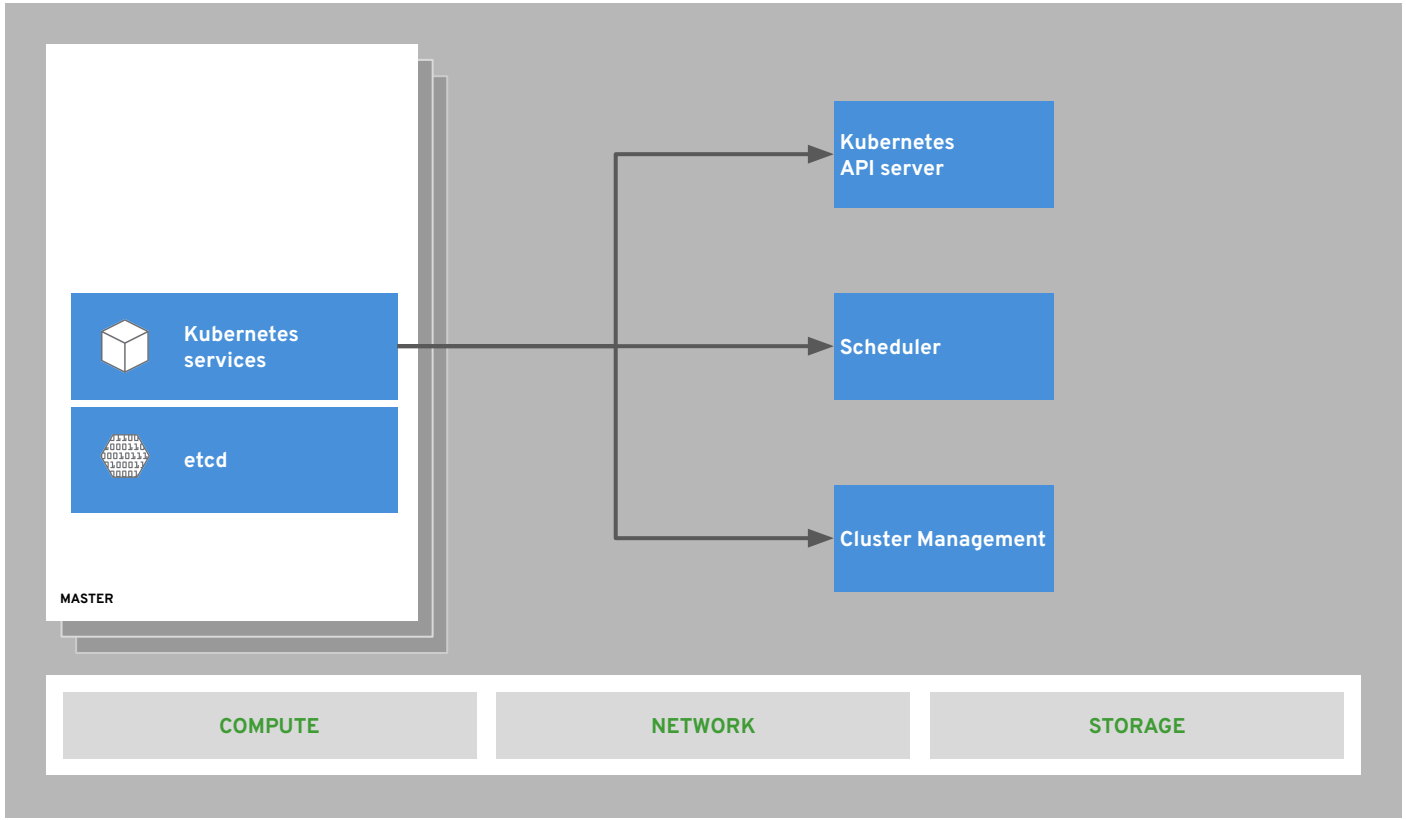
masters are the control plane

everything runs in pods

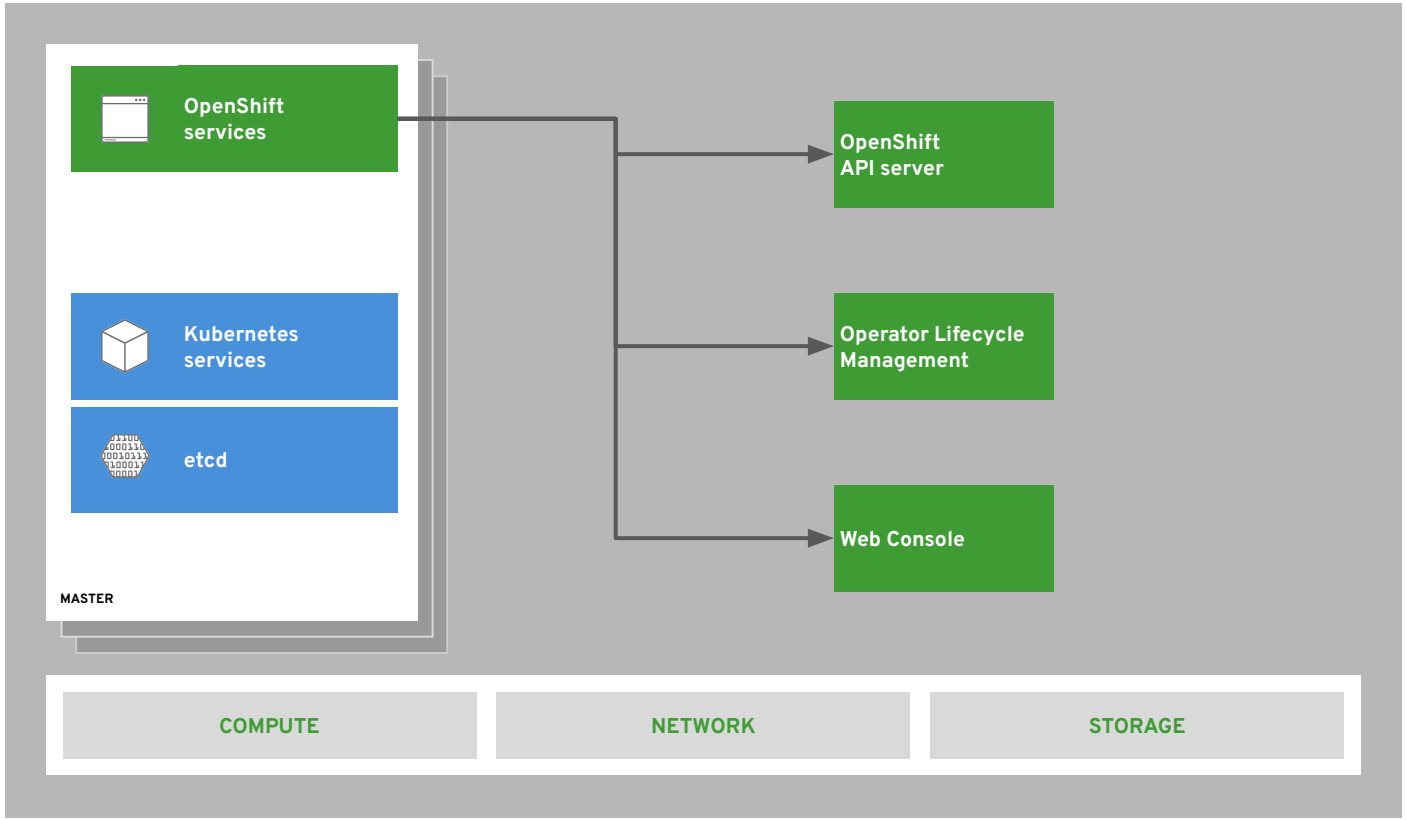


state of everything

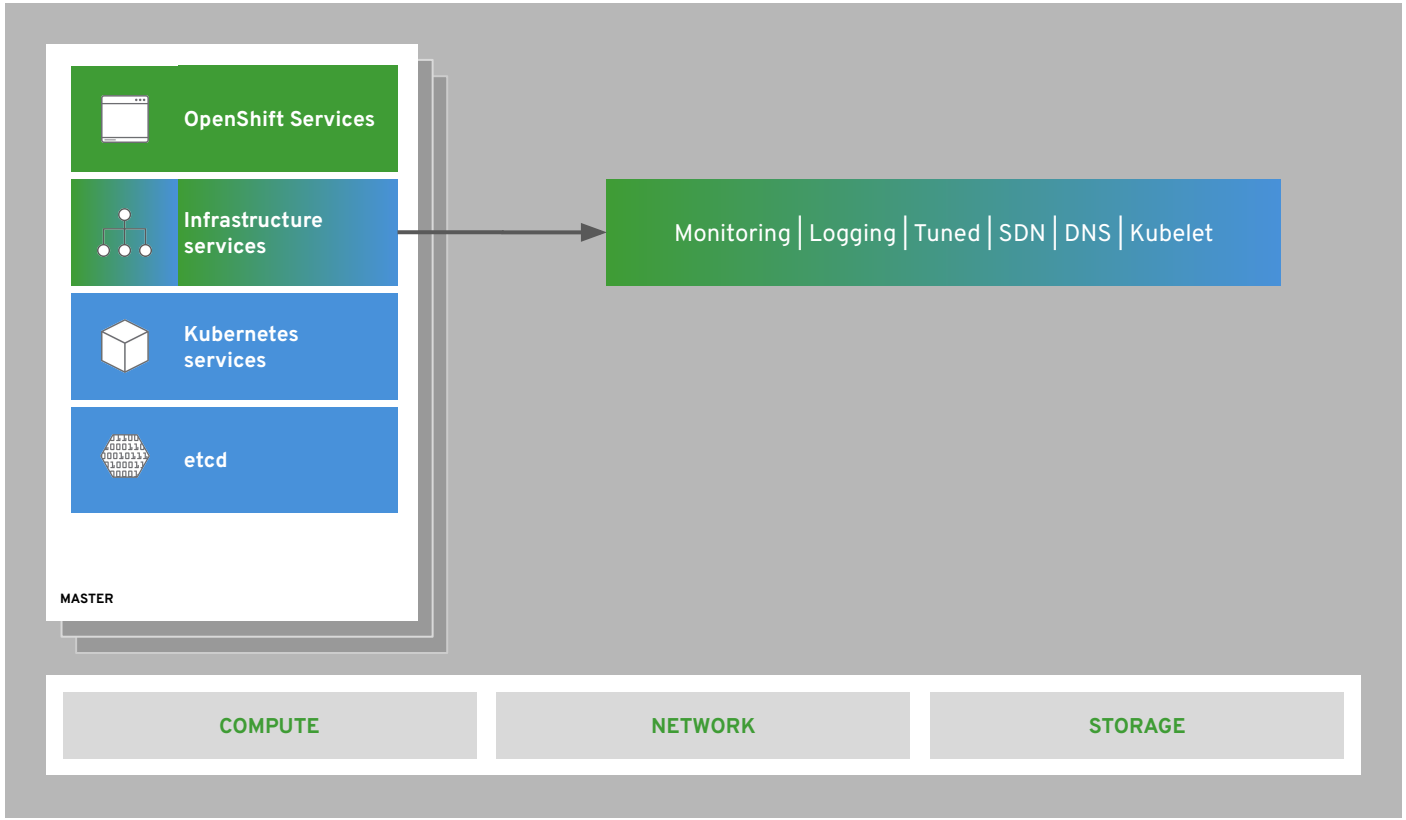
core kubernetes components



core OpenShift components

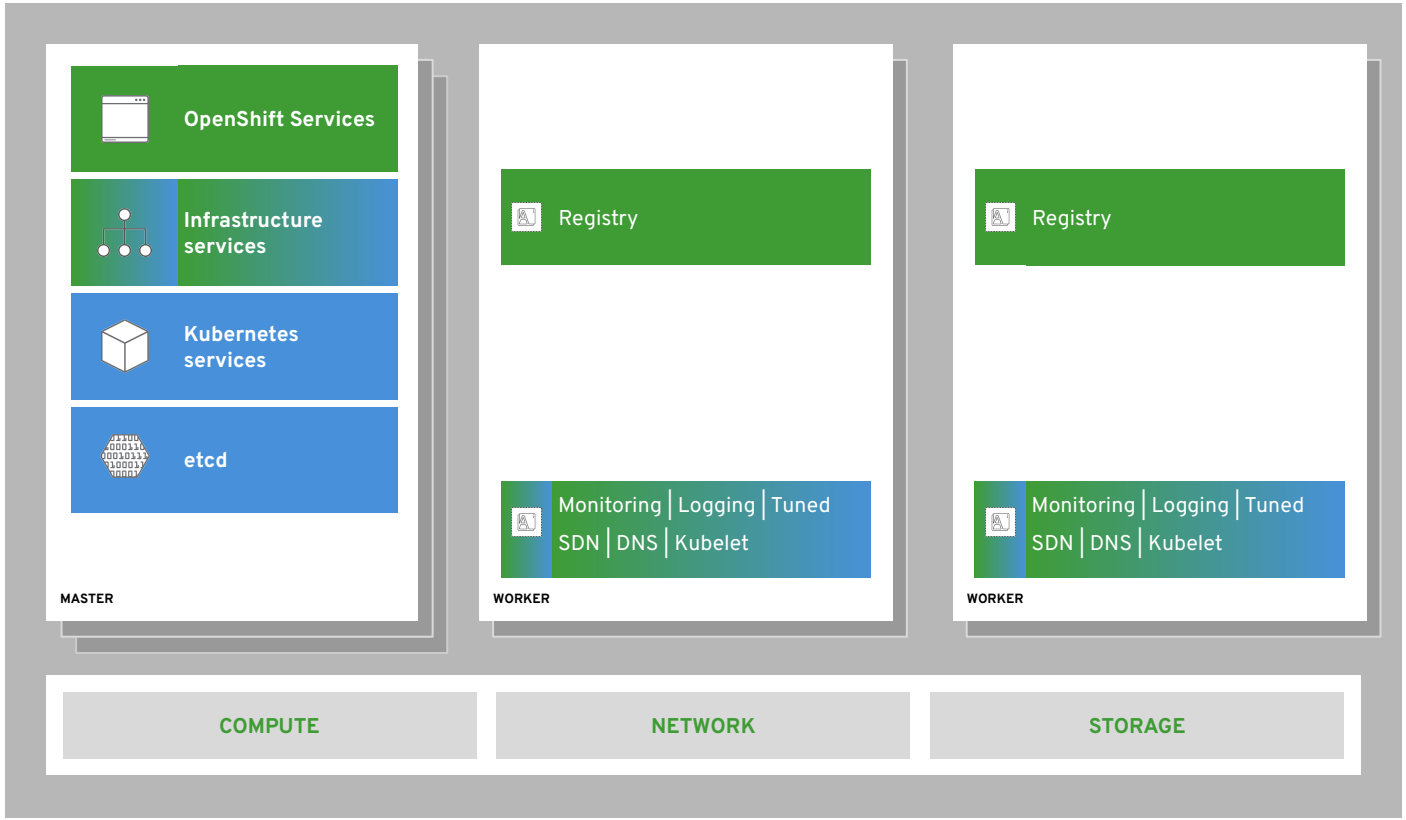


internal and support infrastructure services

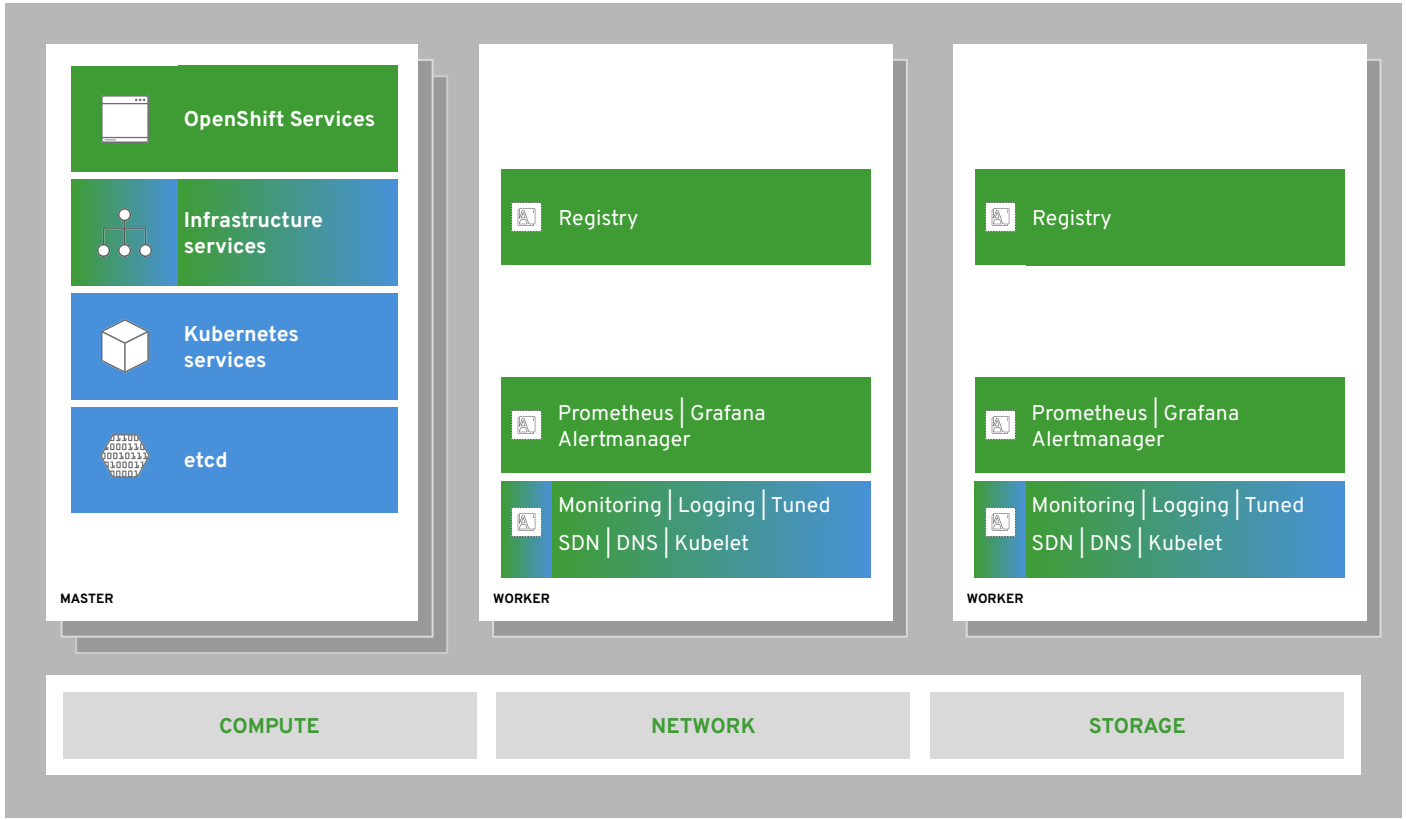


run on all hosts

integrated image registry

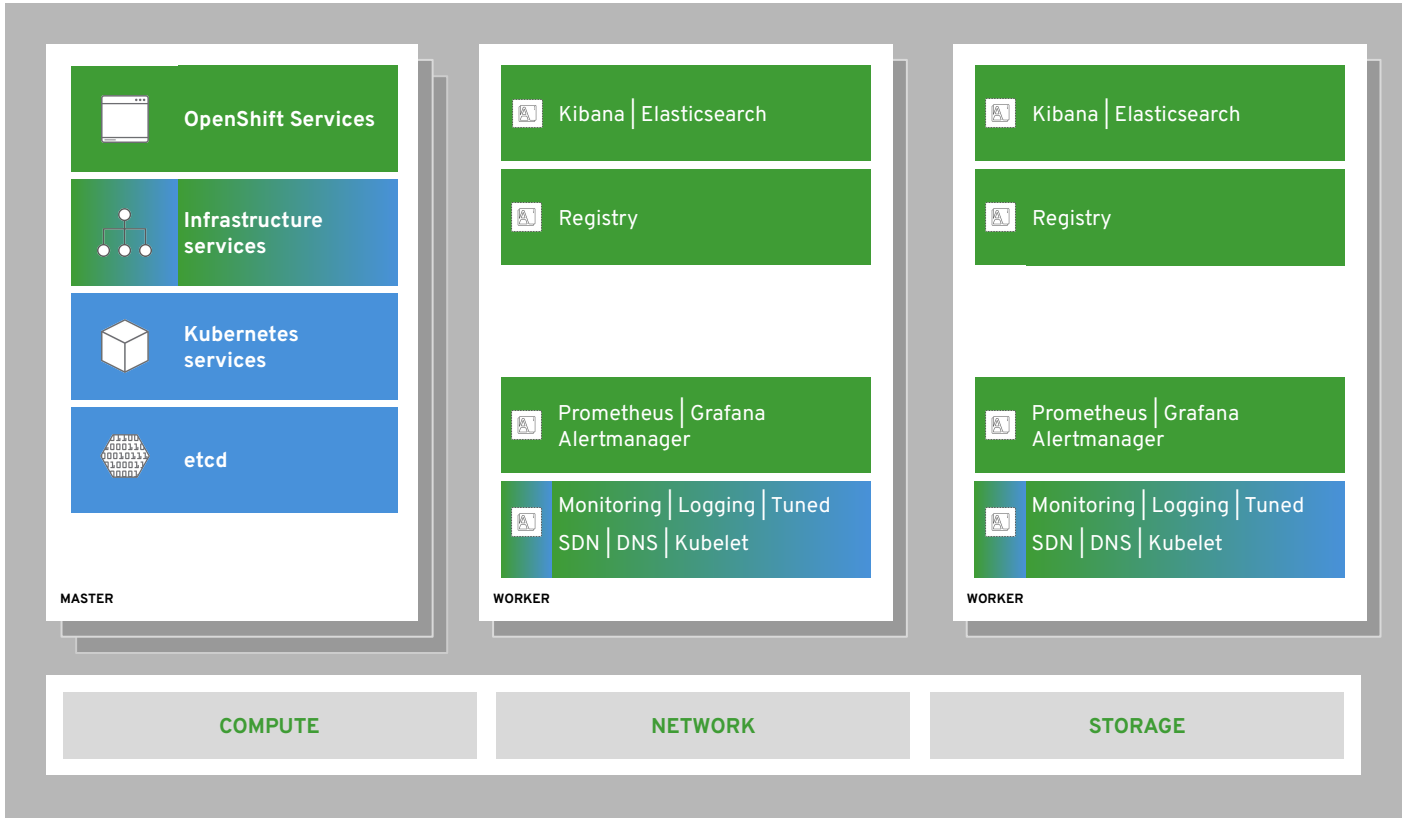


cluster monitoring

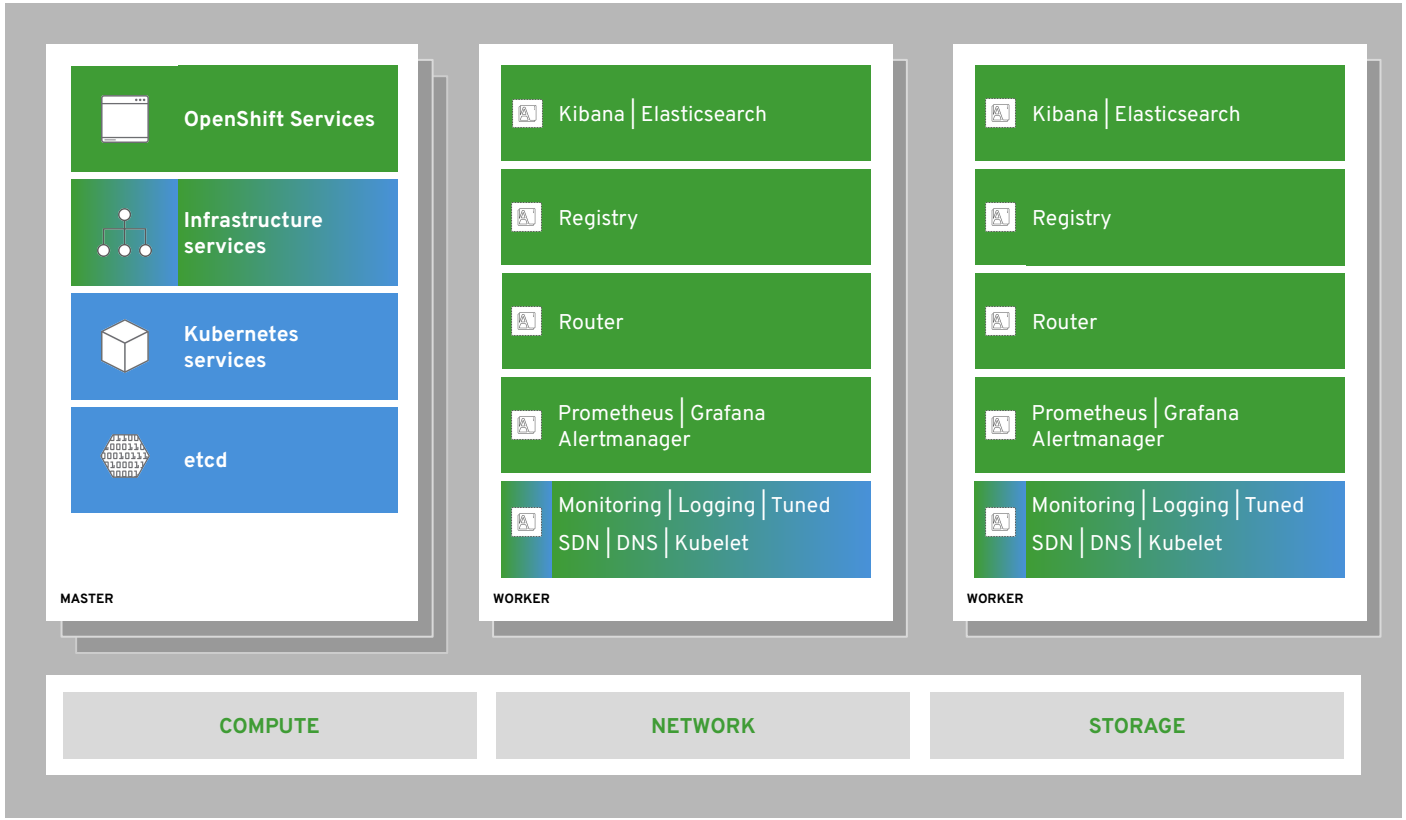


Red Hat

log aggregation

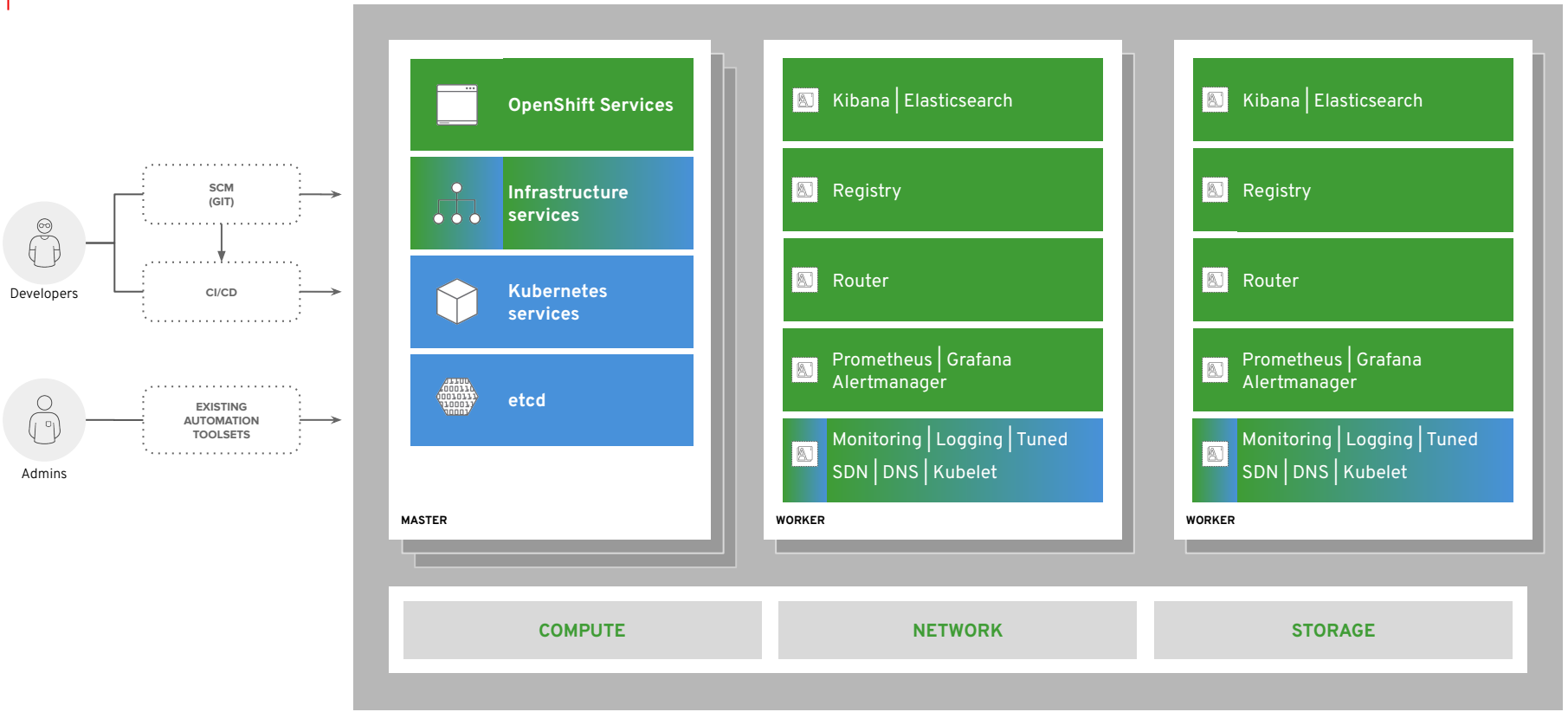


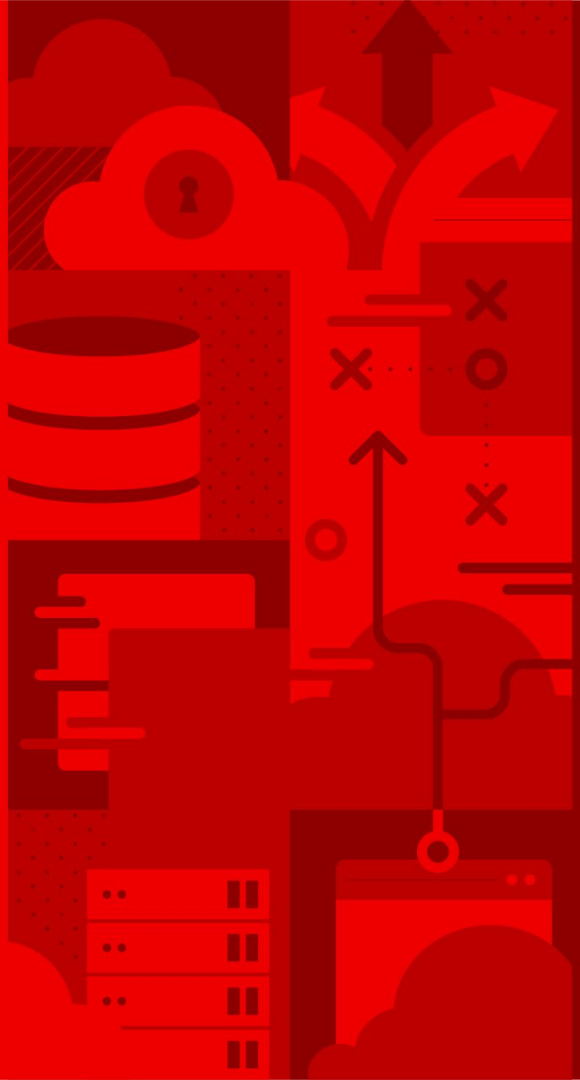
integrated routing



Red Hat

dev and ops via web, cli, API, and IDE





OpenShift lifecycle, installation & upgrades

OpenShift 4 Installation

Two new paradigms

for deploying clusters

Red Hat

Installation Paradigms

OPENSHIFT CONTAINER PLATFORM

Full Stack Automated

Simplified opinionated "Best Practices" for cluster provisioning

Fully automated installation and updates including host container OS.

Red Hat Enterprise Linux CoreOS

Pre-existing Infrastructure

Customer managed resources & infrastructure provisioning

Plug into existing DNS and security boundaries

Red Hat Enterprise Linux CoreOS

Red Hat Enterprise Linux

HOSTED OPENSHIFT

Azure Red Hat OpenShift

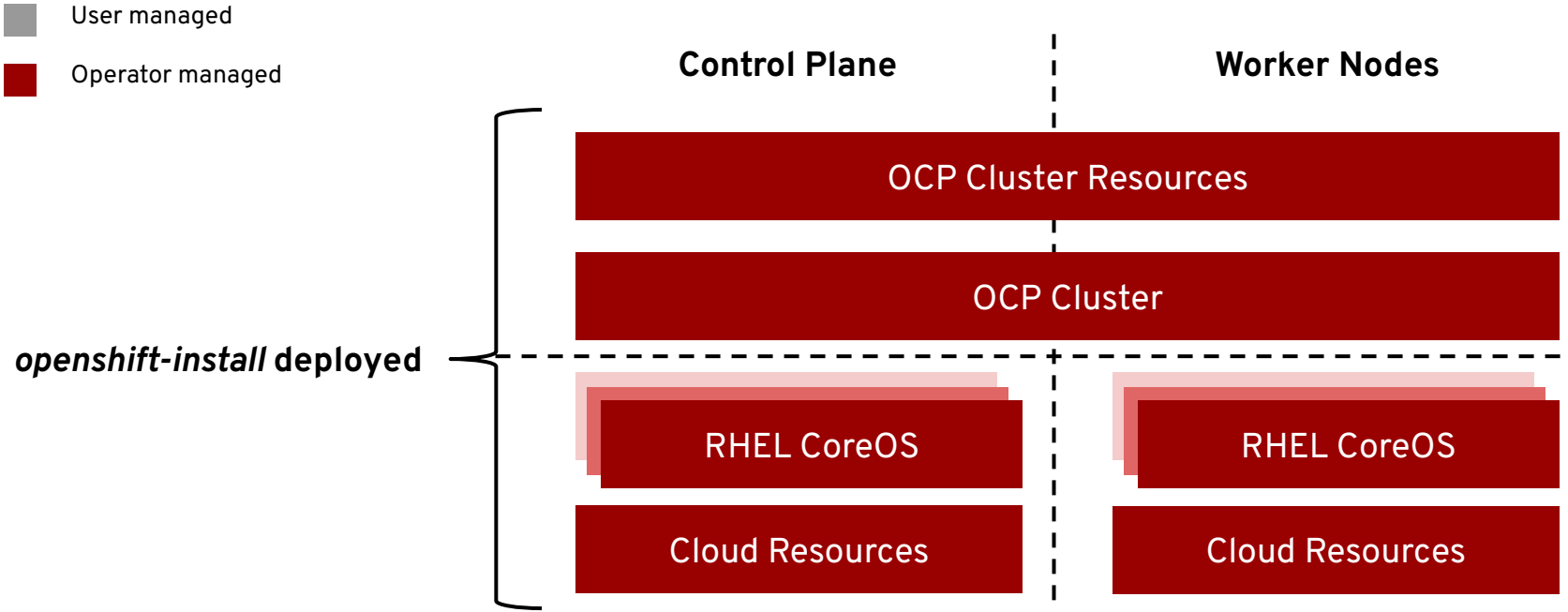
Deploy directly from the Azure console. Jointly managed by Red Hat and Microsoft Azure engineers.

OpenShift Dedicated

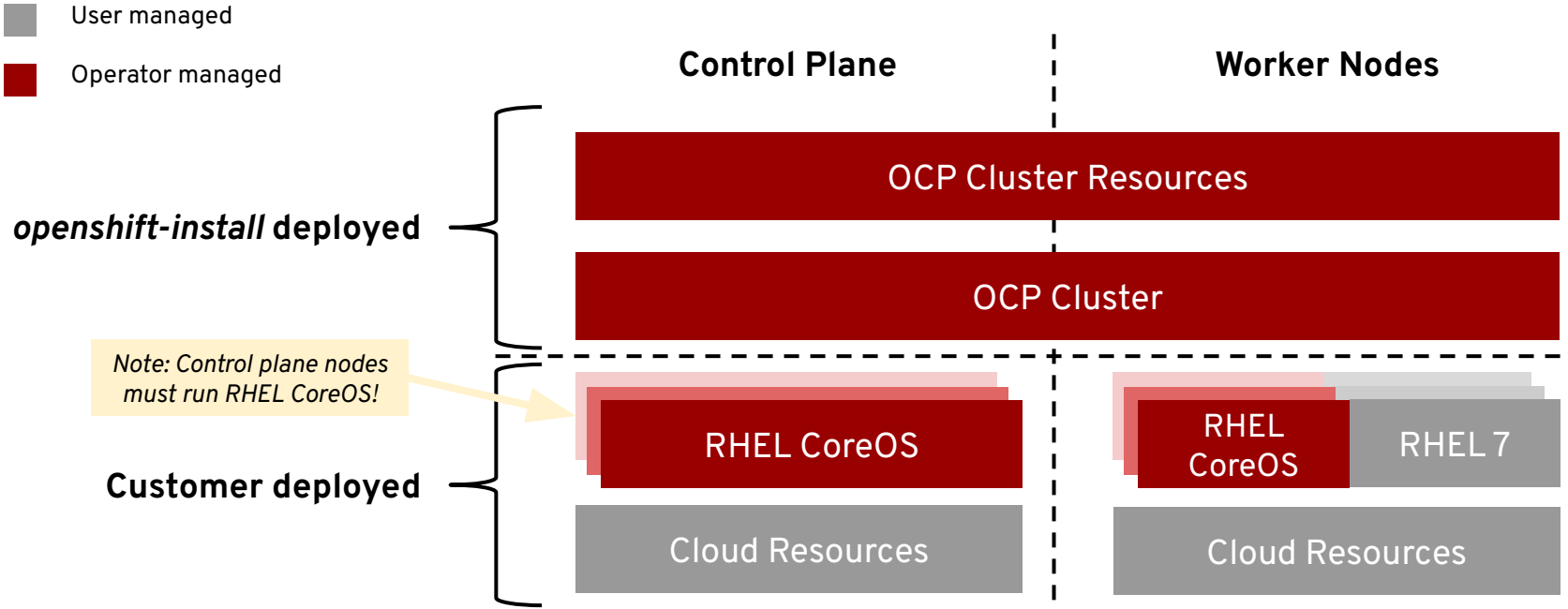
Get a powerful cluster, fully Managed by Red Hat engineers and support.

Red Hat

Full-stack Automated Installation



Pre-existing Infrastructure Installation



Comparison of Paradigms

	Full Stack Automation	Pre-existing Infrastructure
Build Network	Installer	User
Setup Load Balancers	Installer	User
Configure DNS	Installer	User
Hardware/VM Provisioning	Installer	User
OS Installation	Installer	User
Generate Ignition Configs	Installer	Installer
OS Support	Installer: RHEL CoreOS	User: RHEL CoreOS + RHEL 7
Node Provisioning / Autoscaling	Yes	Only for providers with OpenShift Machine API support

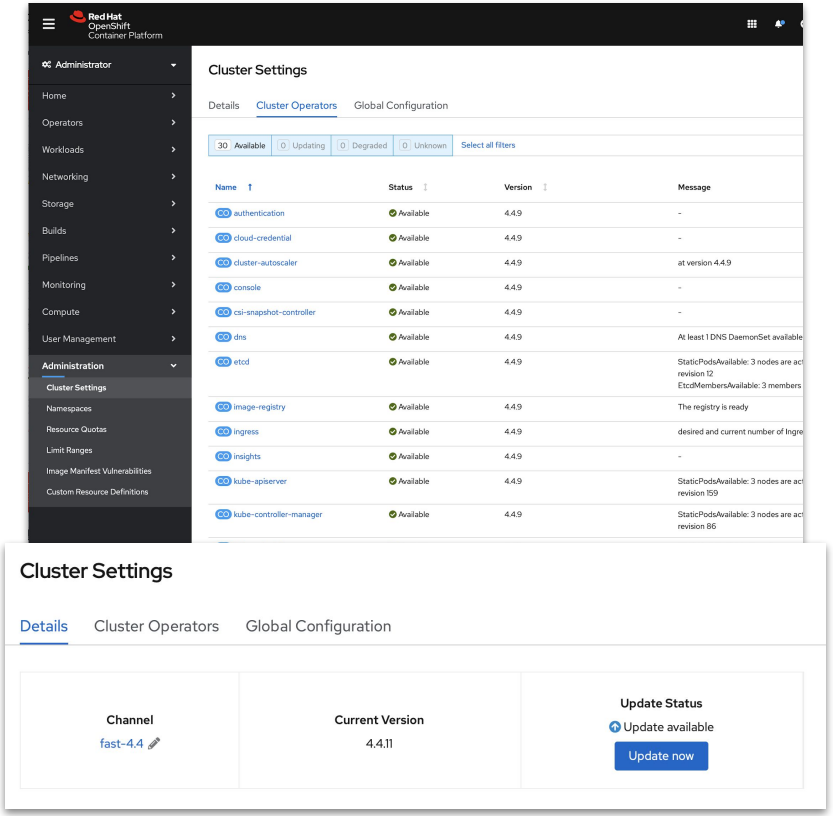
OpenShift 4 Lifecycle

Supported paths for upgrades and migrations

Red Hat

Each OpenShift release is a collection of Operators

- 100% automated, in-place upgrade process
- 30 Operators run every major part of the platform:
  - Console, Monitoring, Authentication, Machine management, Kubernetes Control Plane, etcd, DNS, and more.
- Operators constantly strive to meet the desired state, merging admin config and Red Hat recommendations
- CI testing is constantly running install, upgrade and stress tests against groups of Operators



Red Hat

OpenShift Upgrades and Migrations

Happy path = upgrade through each version

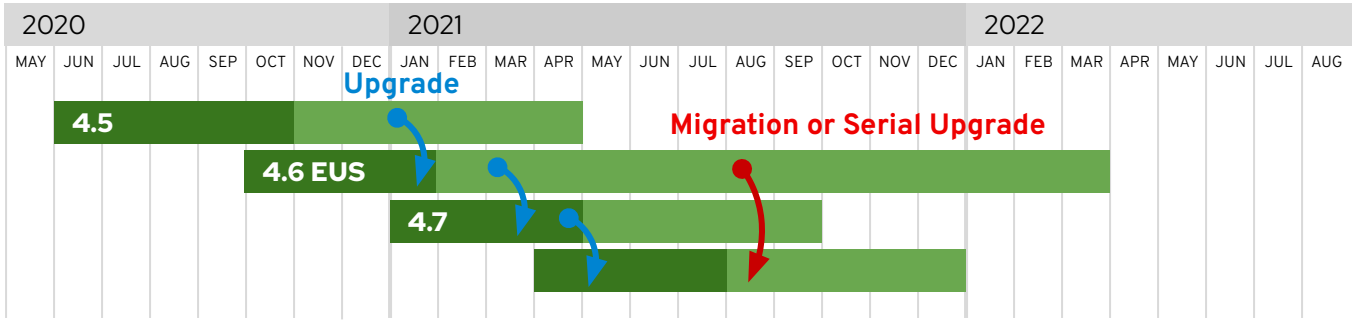
- On a regular cadence, upgrade to the next supported version.

Optional path = migration tooling

- To skip versions or catch up, use the application migration tooling to move to a new cluster.

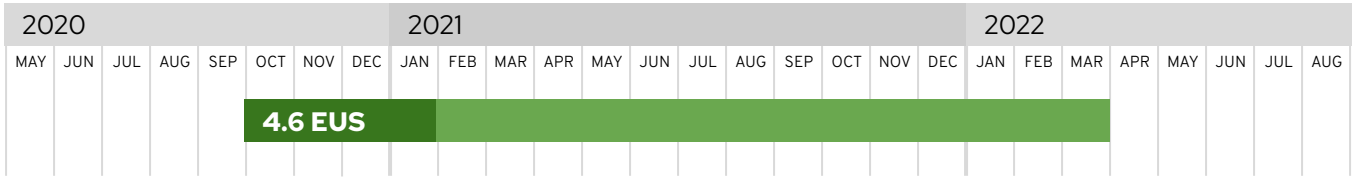
What is Extended Update Support (EUS) ?

- Extended timeframe for critical security and bug fixes
- Work within a customer's release management philosophies
- Goal to provide a serial pathway to update from EUS to EUS
  - Augmented by Migration Tool and/or Advanced Cluster Management (ACM) based on use-case



Red Hat

4.6 EUS for Layered Products/Add-ons



Complete "hands off" EUS

Remain on single supported version for the entire EUS period

OpenShift Logging
OpenShift Container Storage
Advanced Cluster Manager
Process Automation
OpenShift CNF
Jaeger

Mid-cycle refresh during EUS

The EUS cycles for these products refresh during the OpenShift EUS

LAYERED PRODUCT UPGRADE

Cluster Migration Tool
Red Hat SSO
JBoss EAP
Quarkus
Thorntail
Spring Boot
Vert.x
JWS (Tomcat)
DataGrid

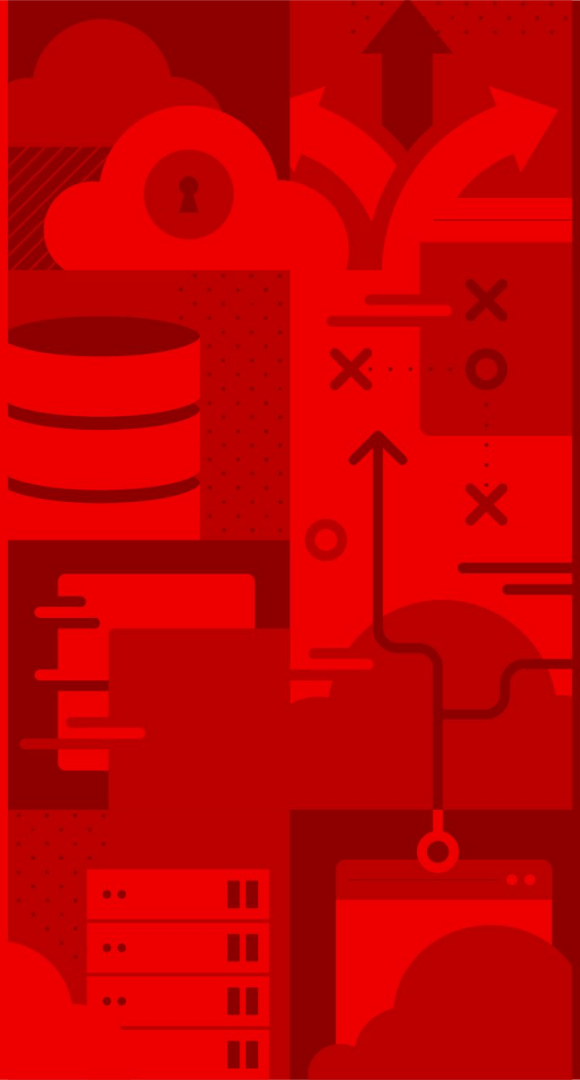
Normal updates during EUS

Follows the normal support window for the add-on, shorter than EUS

LAYERED UPGRADE LAYERED UPGRADE LAYERED UPGRADE LAYERED UPGRADE LAYERED UPGRADE

OpenShift Virtualization
OpenShift Serverless
OpenShift Pipelines
OpenShift Service Mesh
CodeReady Containers
Red Hat Quay / CSO

Red Hat



Operations and infrastructure deep dive

Red Hat

Red Hat Enterprise Linux CoreOS

The OpenShift operating system and its runtime components

Red Hat

Red Hat Enterprise Linux

	RED HAT® ENTERPRISE LINUX®	RED HAT® ENTERPRISE LINUX CoreOS
	General Purpose OS	**Immutable container host**
BENEFITS	• 10+ year enterprise life cycle • Industry standard security • High performance on any infrastructure • Customizable and compatible with wide ecosystem of partner solutions	• Self-managing, over-the-air updates • Immutable and tightly integrated with OpenShift • Host isolation is enforced via Containers • Optimized performance on popular infrastructure
WHEN TO USE	When customization and integration with additional solutions is required	When cloud-native, hands-free operations are a top priority

Immutable Operating System

Red Hat Enterprise Linux CoreOS is versioned with OpenShift

CoreOS is tested and shipped in conjunction with the platform. Red Hat runs thousands of tests against these configurations.

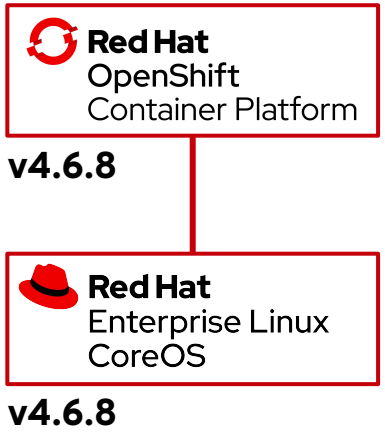
Red Hat Enterprise Linux CoreOS is managed by the cluster

The Operating system is operated as part of the cluster, with the config for components managed by Machine Config Operator:

- CRI-O config
- Kubelet config
- Authorized registries
- SSH config

RHEL CoreOS admins are responsible for:

Nothing. 😀🙌



Runtime, Build, Synchronize

OCI tooling to create, run, and manage, Linux Containers with a cluster-friendly life cycle

A lightweight OCI-compliant runtime

- Minimal and secure architecture
- Optimized for Kubernetes
- Run any OCI-compliant container image (including docker)

A docker-compatible CLI for containers

- Remote management API via Varlink
- Image/container tagging
- Advanced namespace isolation

Inspect, push/pull, and sign OCI images

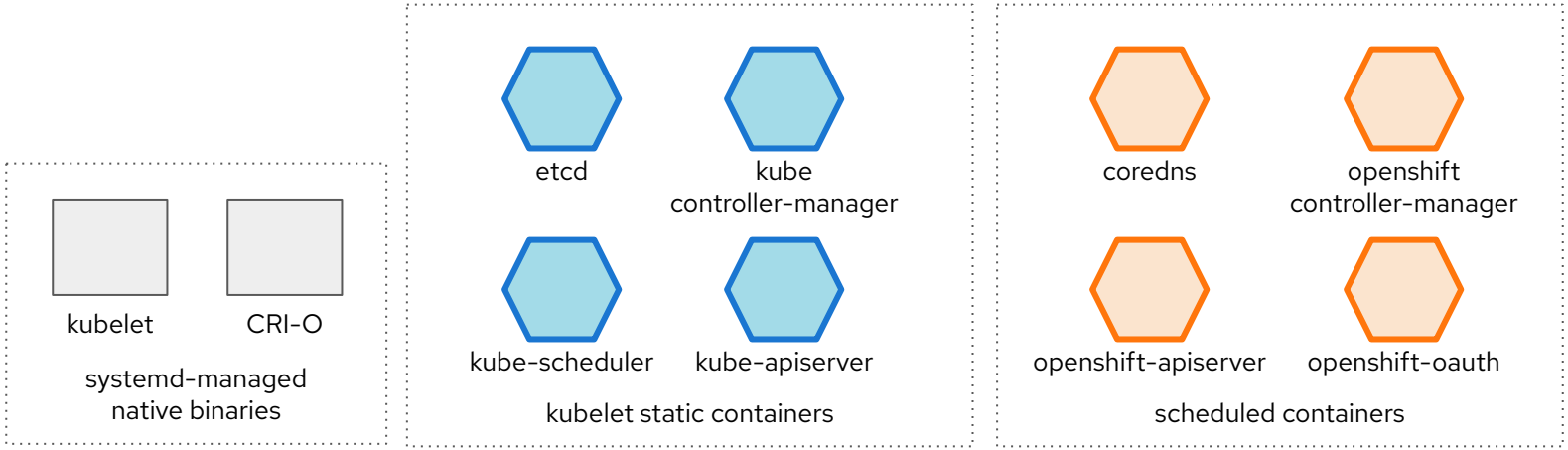
- Inspect image manifests
- Transfer images between registries

CRI-O Support in OpenShift

CRI-O tracks and versions identical to Kubernetes, simplifying support permutations

CRI-O 1.12 	Kubernetes 1.12 	OpenShift 4.0 
CRI-O 1.13 	Kubernetes 1.13 	OpenShift 4.1 
CRI-O 1.14 	Kubernetes 1.14 	OpenShift 4.2 

RHEL CoreOS “pod” architecture



OpenShift 4 installation

Installer and

user-provisioned

infrastructure,

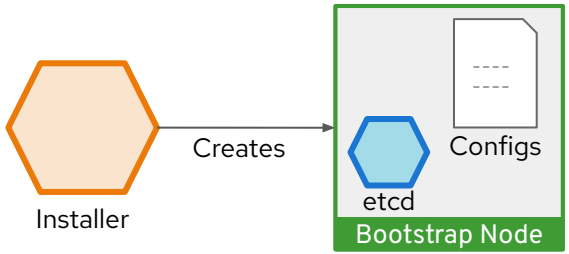
bootstrap, and more

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

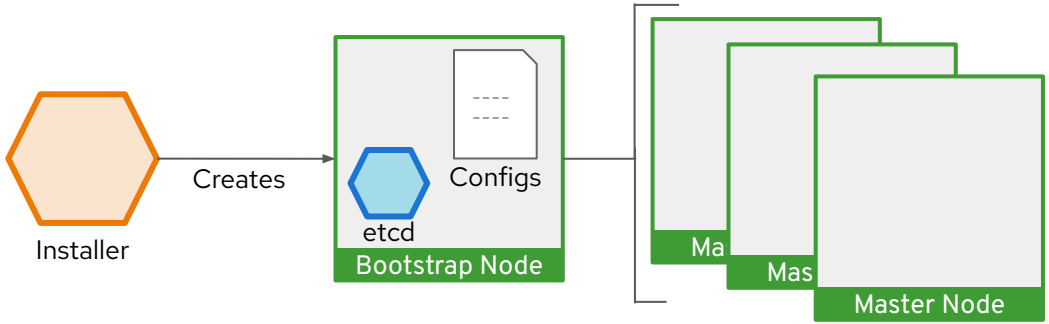
OpenShift Bootstrap Process: Step by Step



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

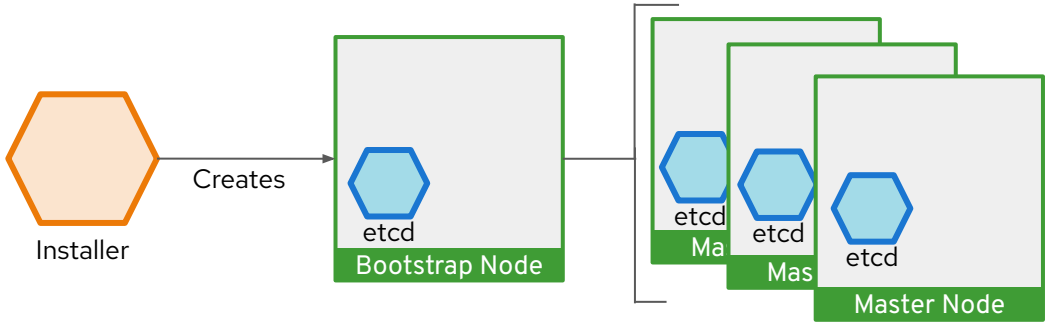
OpenShift Bootstrap Process: Step by Step



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.

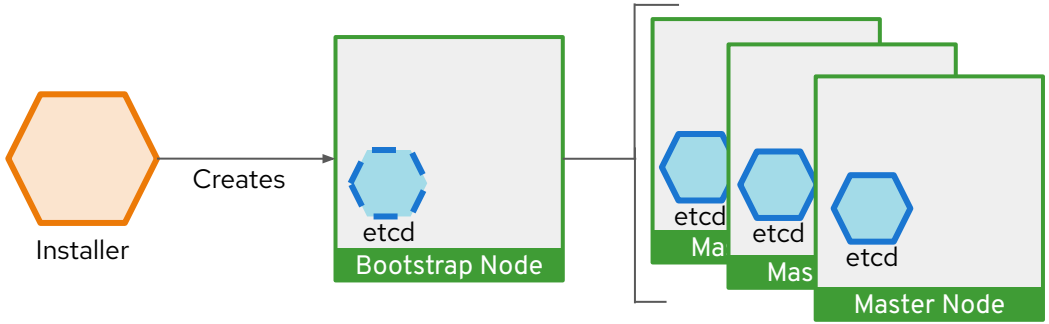
OpenShift Bootstrap Process: Step by Step



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.

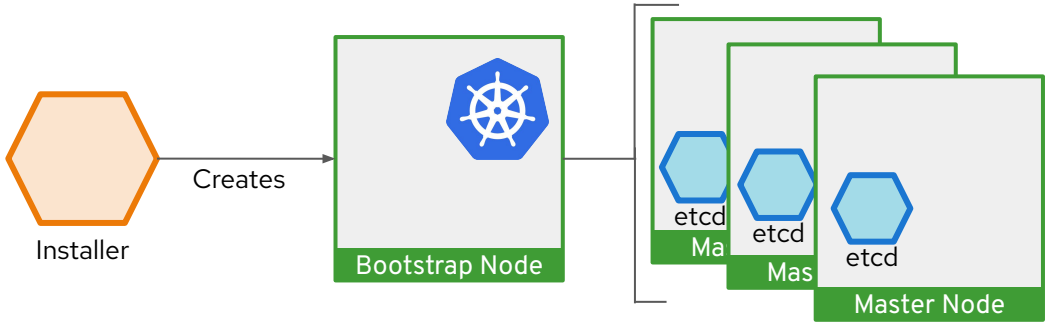
OpenShift Bootstrap Process: Step by Step



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

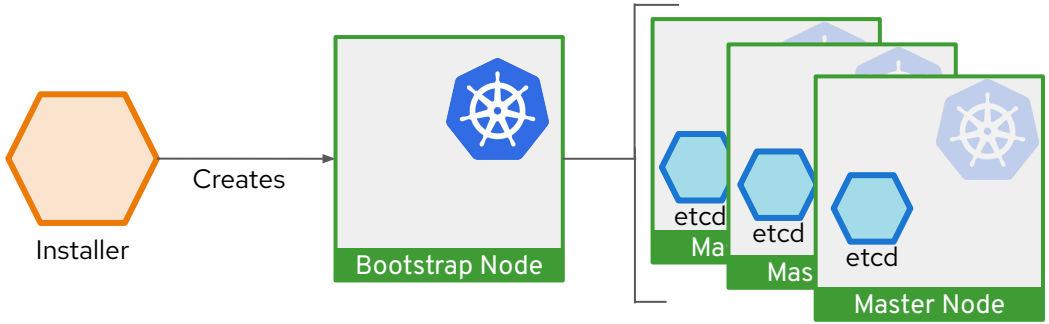
OpenShift Bootstrap Process: Step by Step



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.

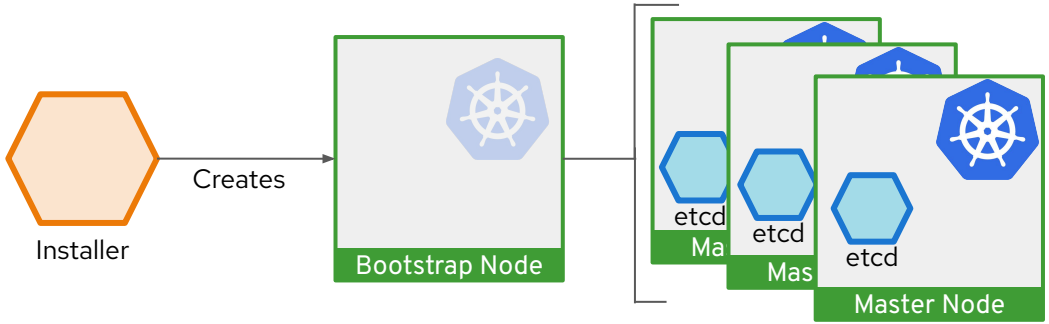
OpenShift Bootstrap Process: Step by Step



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.

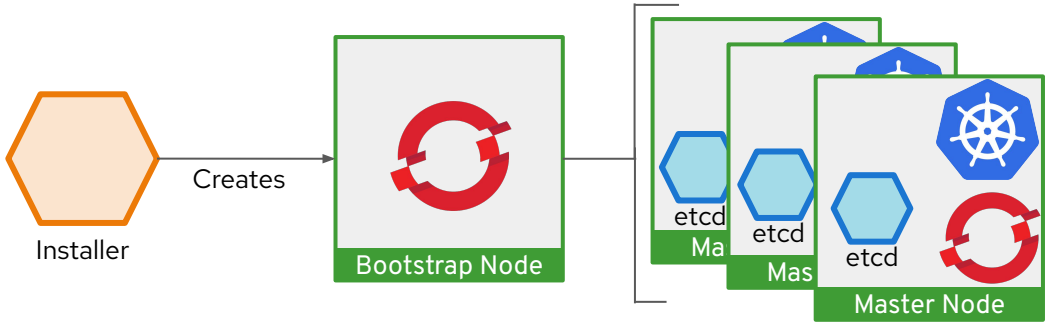
OpenShift Bootstrap Process: Step by Step



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.

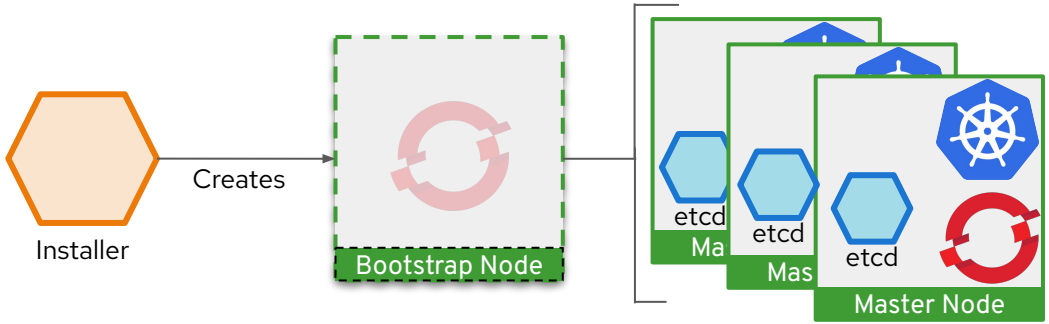
OpenShift Bootstrap Process: Step by Step



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.

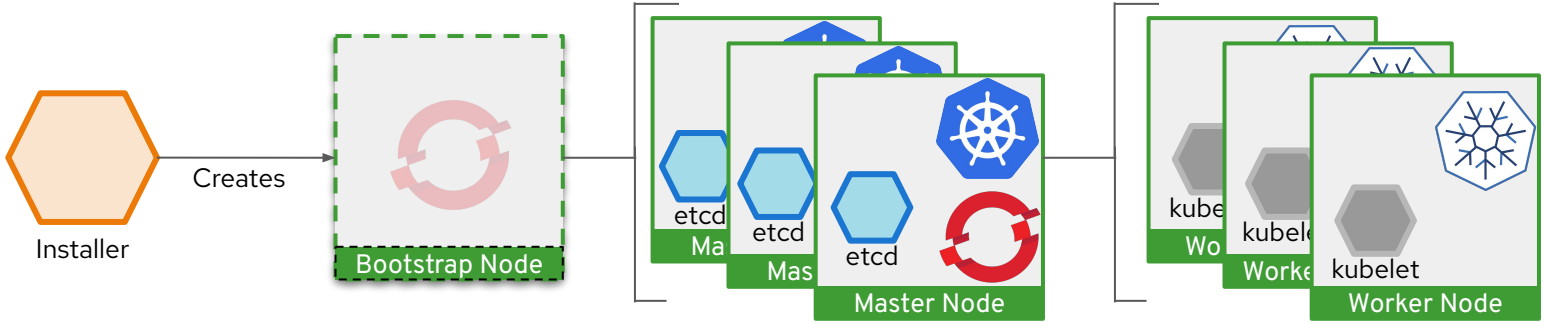
OpenShift Bootstrap Process: Step by Step



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.

How everything deployed comes under management

Masters (Special)

- Full Stack Automated: Installer provisions minimal viable masters
- User Provisioned: User/Administrator provisions minimal viable masters
- Machine API adopts existing masters post-provision
- Each master is a standalone Machine object
- Termination protection (avoid self-destruction)

Workers

- Each Machine Pool corresponds to MachineSet
- Optionally autoscale (min,max) and health check (replace if not ready > X minutes)

Multi-AZ

- MachineSets scoped to single AZ
- Installer stripes N machine sets across AZs by default
- Post-install best effort balance via cluster autoscaler

One Touch provisioning via Ignition

Machine generated; Machine validated

Ignition applies a declarative node configuration early in the boot process. Unifies kickstart and cloud-init.

- Generated via openshift-install
- Configures storage, systemd units, users, & remote configs
- Executed in the initramfs
- Configuration for masters & workers is served from the control plane and sourced from Machine Configs

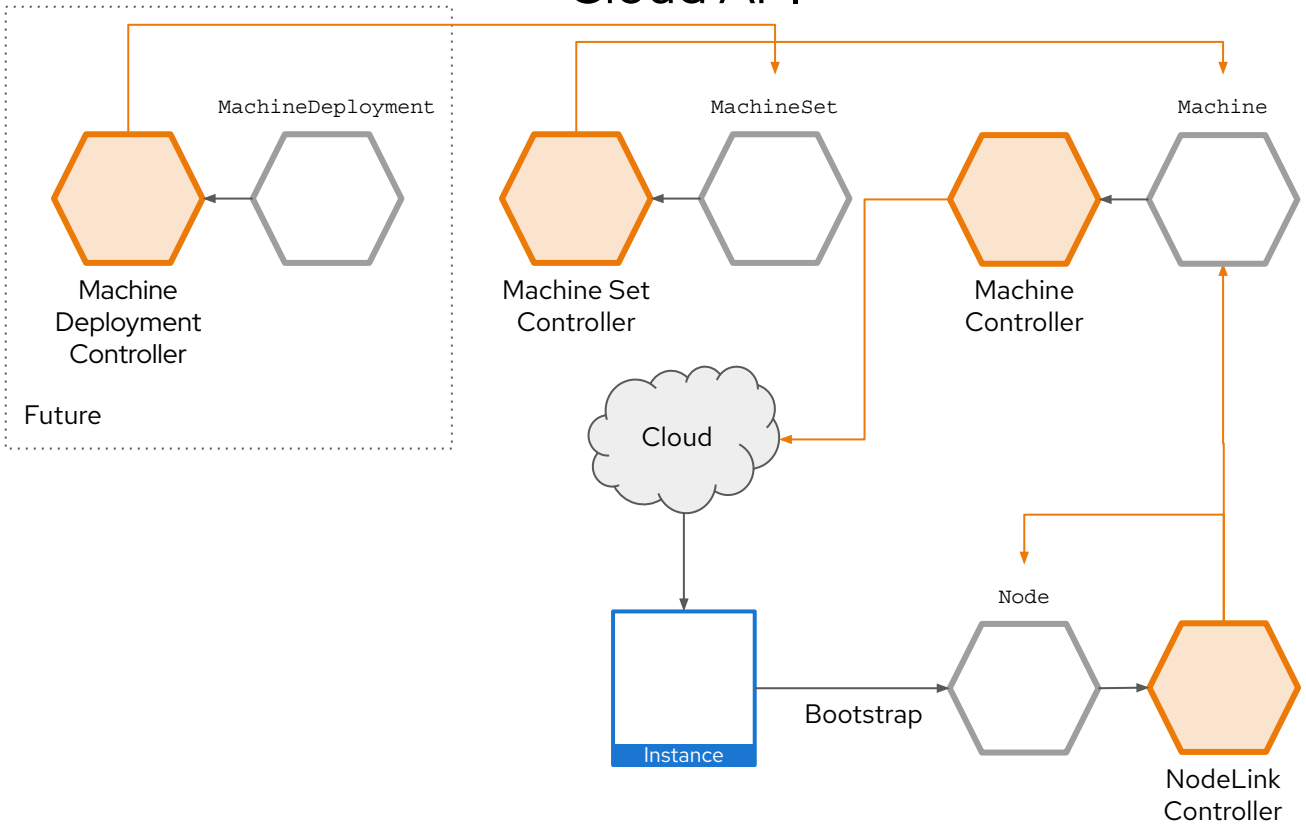
```
{
  "ignition": {
    "config": {},
    "timeouts": {},
    "version": "2.1.0"
  },
  "passwd": {
    "users": [
      {
        "name": "core",
        "passwordHash": "$6$43y3tkl...",
        "sshAuthorizedKeys": [
          "key1"
        ]
      }
    ]
  },
  "storage": {},
  "systemd": {}
}
```

OpenShift 4 Cluster Management

Powered by Operators, OpenShift 4 automates many cluster management activities

Red Hat

Cloud API



Machine Config Operator

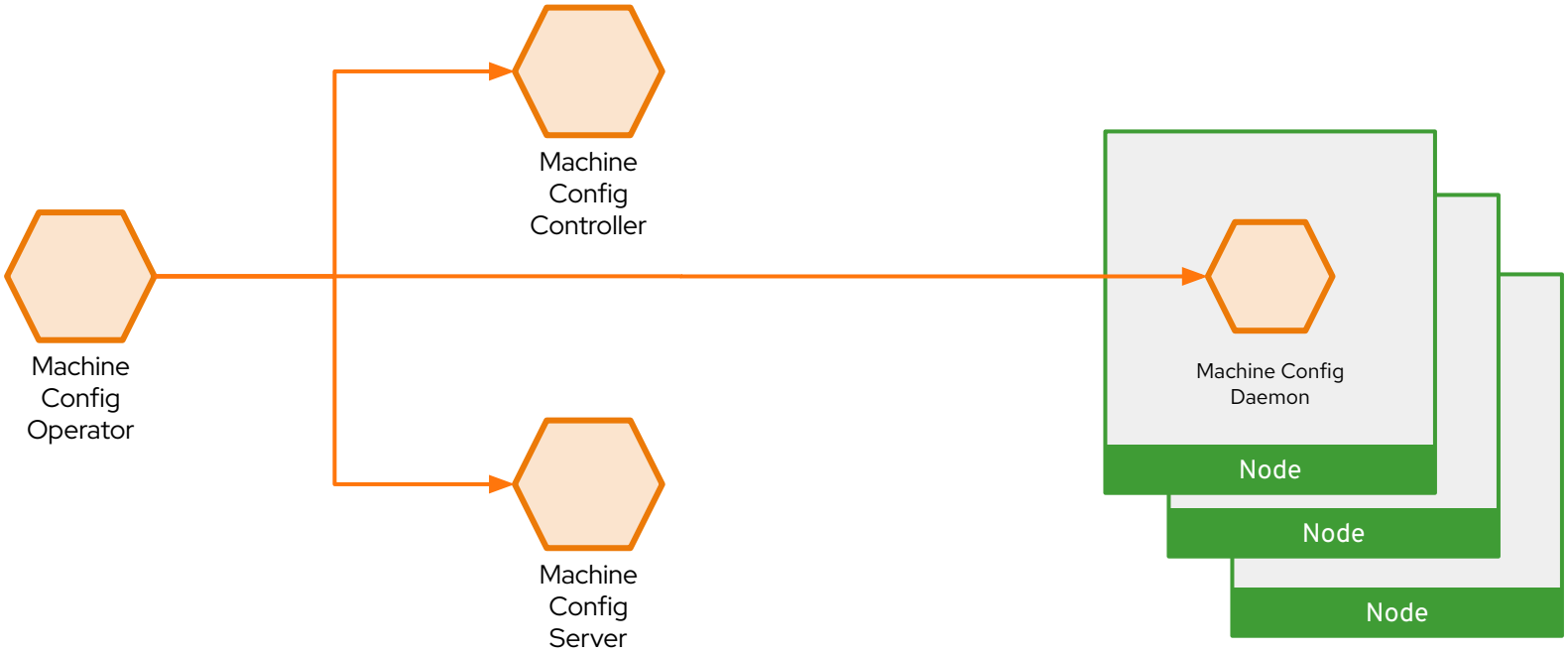
A Kube-native way to configure hosts

OS configuration is stored and applied across the cluster via the Machine Config Operator.

- Subset of ignition modules applicable post provisioning
  - SSH keys
  - Files
  - systemd units
  - kernel arguments
- Standard k8s YAML/JSON manifests
- Desired state of nodes is checked/fixed regularly
- Can be paused to suspend operations

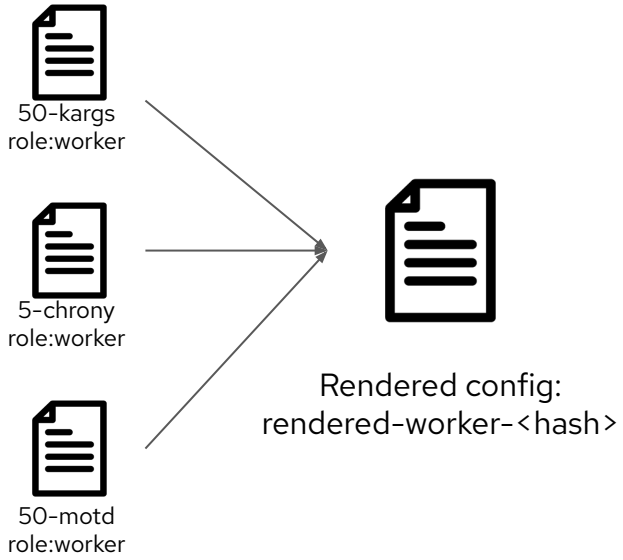
```yaml
# test.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: test-file
spec:
  config:
    storage:
      files:
      - contents:
          source: data:,hello%20world%0A
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/test
```

Operator/Operand Relationships



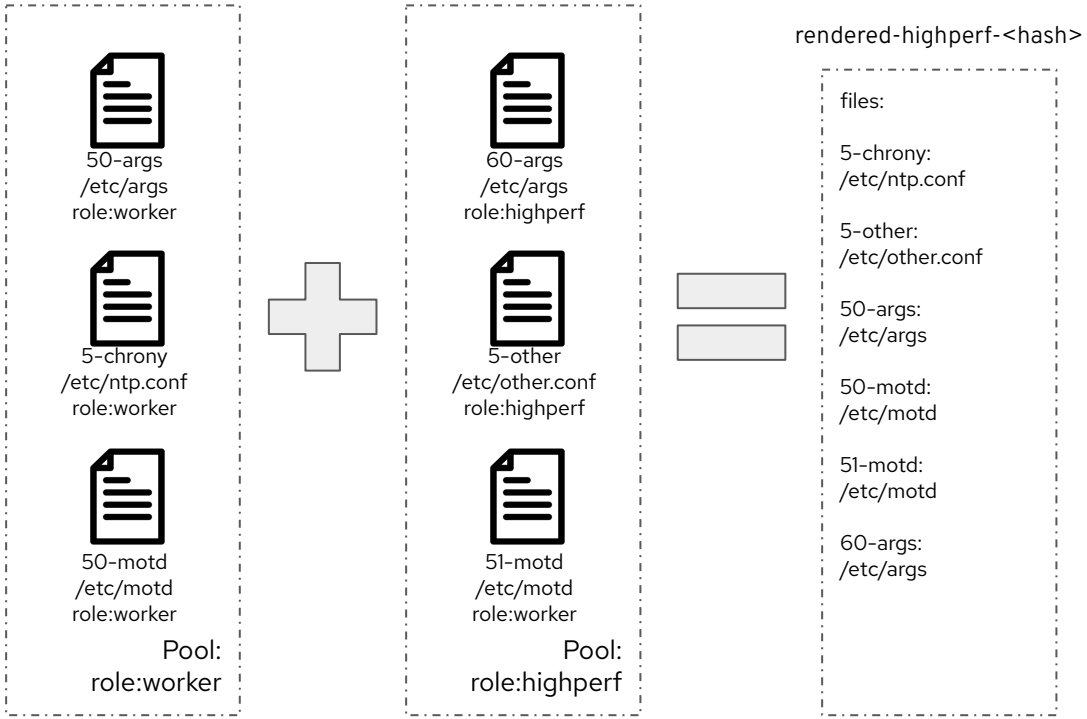
Machine Config and Machine Config Pool

Inheritance-based mapping of configuration to nodes



Custom Machine Config Pools

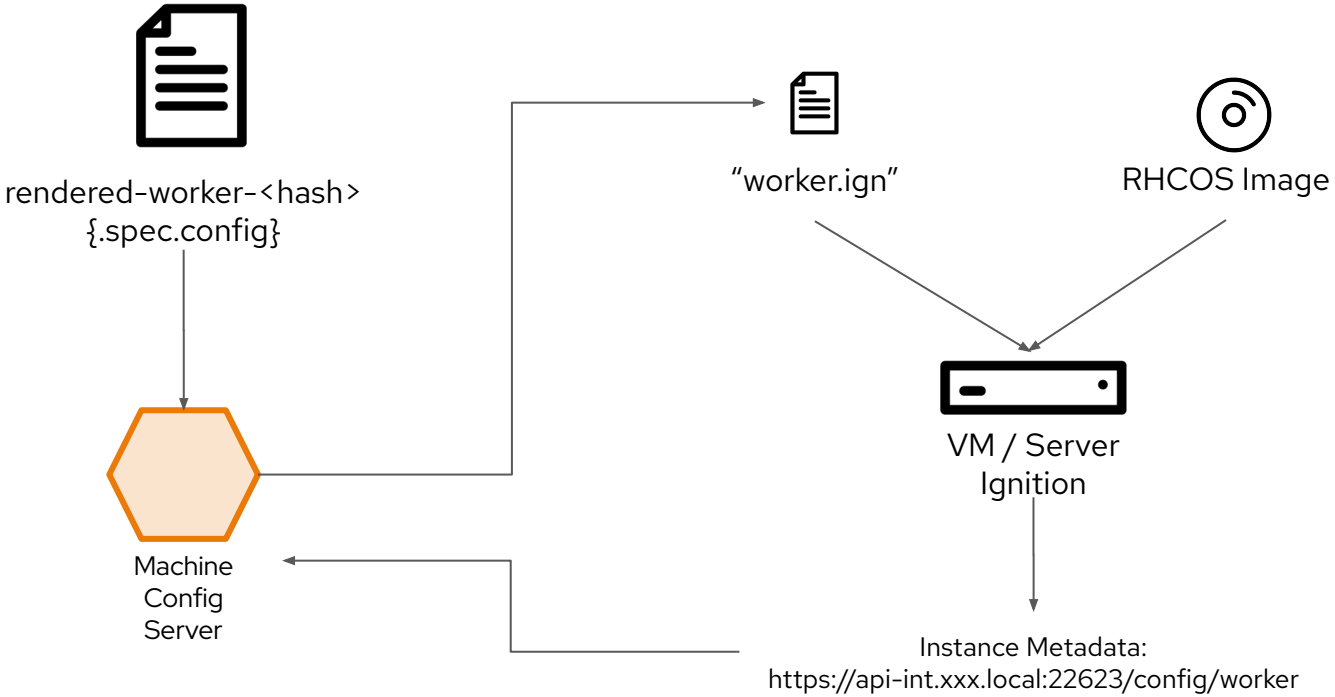
Hierarchical/layered configuration rendering



Red Hat

Machine Config Server

Providing Ignition configuration for provisioning



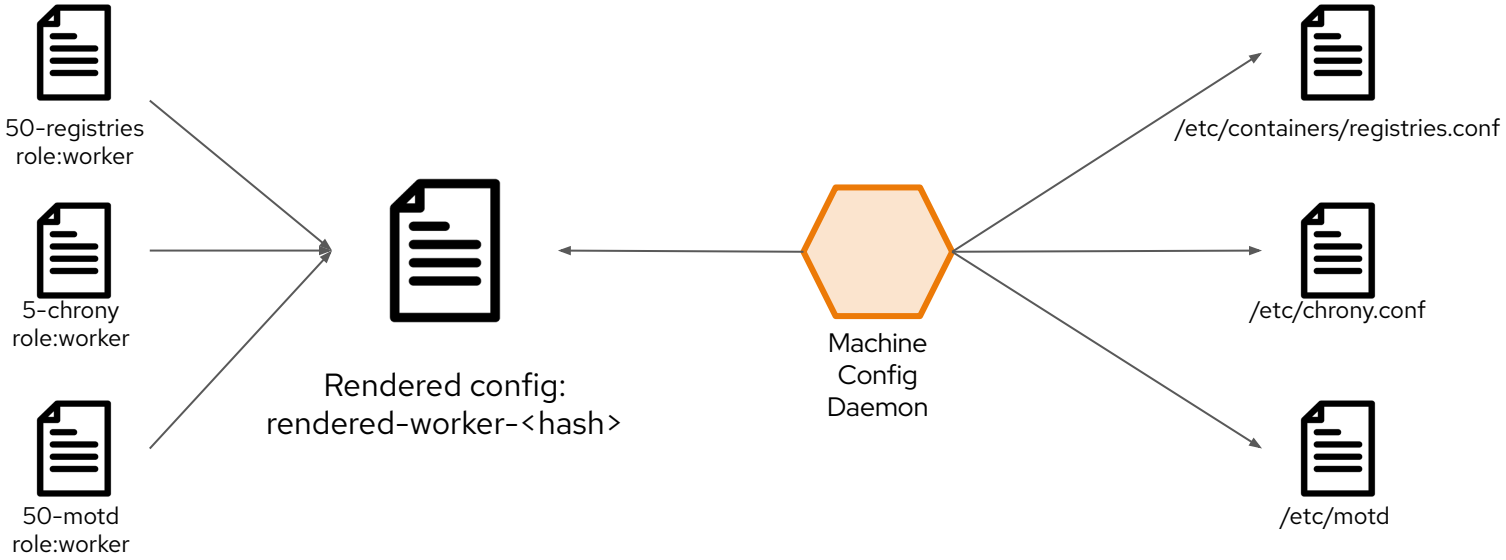
Machine Config Server

Identical nodes at massive scale

Red Hat

Machine Config Daemon

Preventing drift



Red Hat

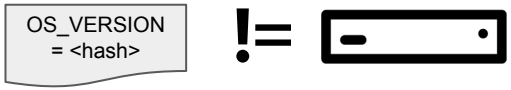
Machine Config Daemon

Acting on drift

The MCO coordinates with the MCD to perform the following actions, in a rolling manner, when OS updates and/or configuration changes are applied:

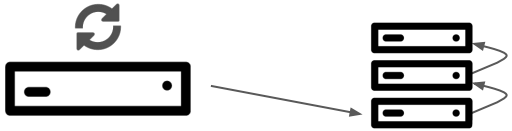
- Cordon / uncordons nodes
- Drain pods
- Stage node changes
  - OS upgrade
  - config changes
  - systemd units
- Reboot

1. Validates node state matches desired state



2. Validate cluster state & policy to apply change

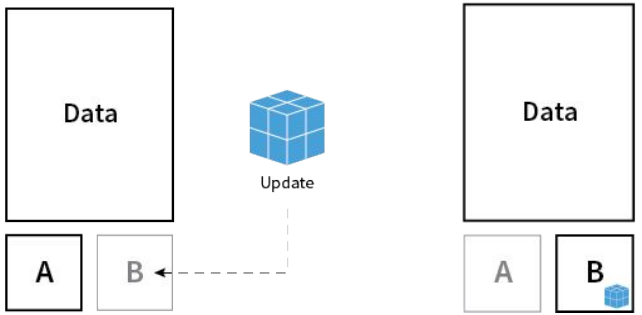
3. Change is rolled across cluster



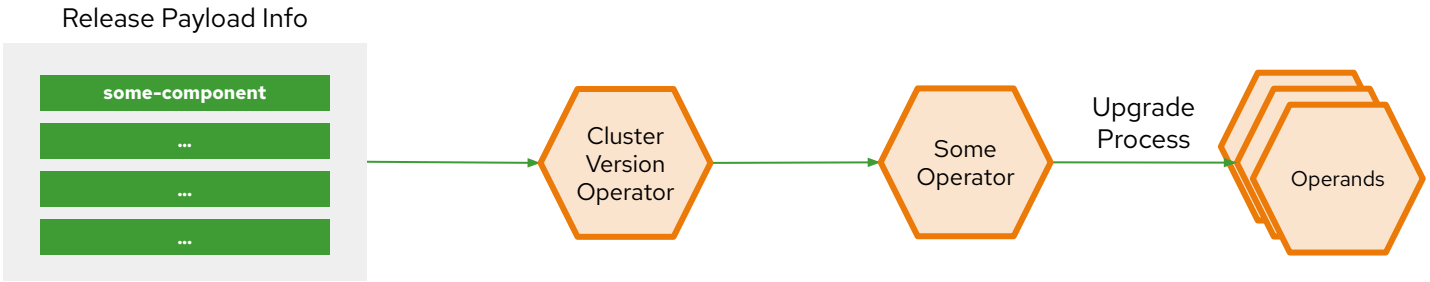
Transactional updates with rpm-ostree

Transactional updates ensure that RHEL CoreOS is never altered during runtime. Rather it is booted directly into an always “known good” version.

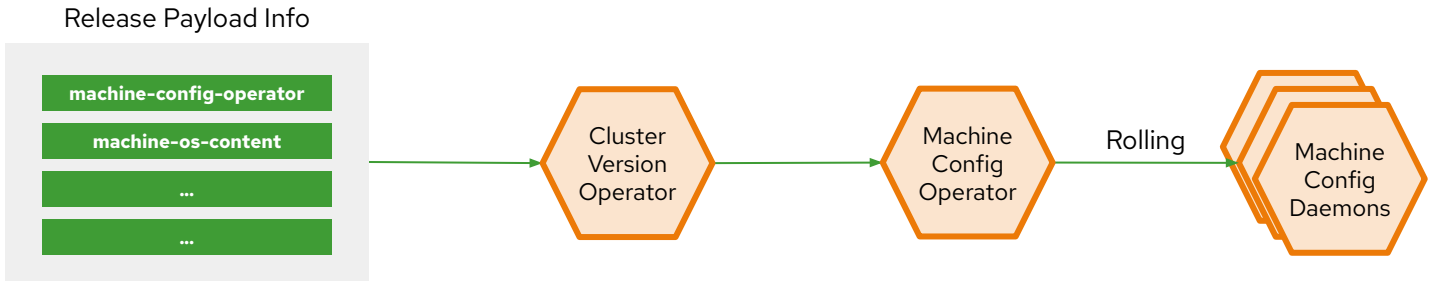
- Each OS update is versioned and tested as a complete image.
- OS binaries (/usr) are read-only
- OS updates encapsulated in container images
- file system and package layering available for hotfixes and debugging

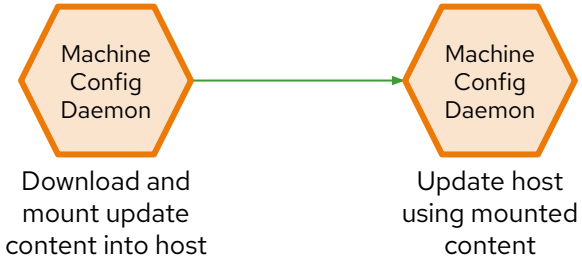


Over-the-air updates: Cluster Components



Over-the-air updates: Nodes





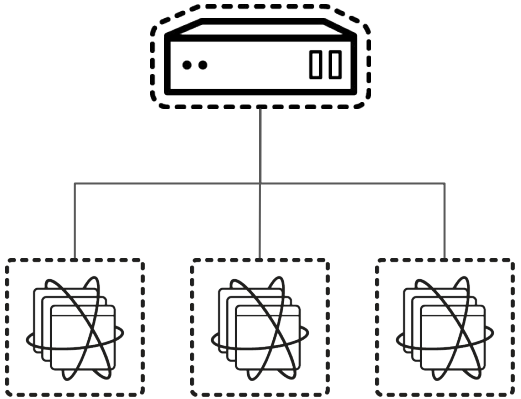
routes and ingress

How traffic enters the cluster

Red Hat

Routing and Load Balancing

- Pluggable routing architecture
  - HAProxy Router
  - F5 Router
- Multiple-routers with traffic sharding
- Router supported protocols
  - HTTP/HTTPS
  - WebSockets
  - TLS with SNI
- Non-standard ports via cloud load-balancers, external IP, and NodePort

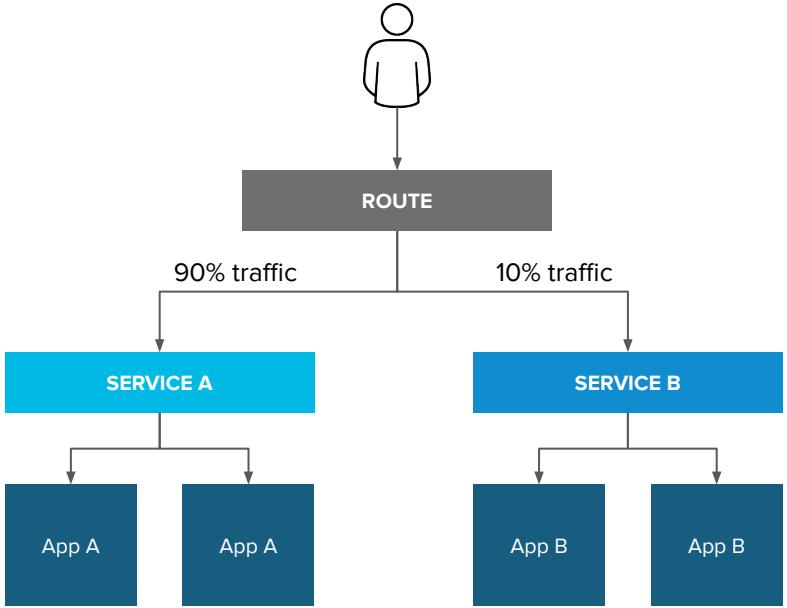


Routes vs Ingress

Feature	Ingress	Route
Standard Kubernetes object	X	
External access to services	X	X
Persistent (sticky) sessions	X	X
Load-balancing strategies (e.g. round robin)	X	X
Rate-limit and throttling	X	X
IP whitelisting	X	X
TLS edge termination	X	X
TLS re-encryption	X	X
TLS passthrough	X	X
Multiple weighted backends (split traffic)		X
Generated pattern-based hostnames		X
Wildcard domains		X

Router-based deployment methodologies

Split Traffic Between Multiple Services For A/B Testing, Blue/Green and Canary Deployments

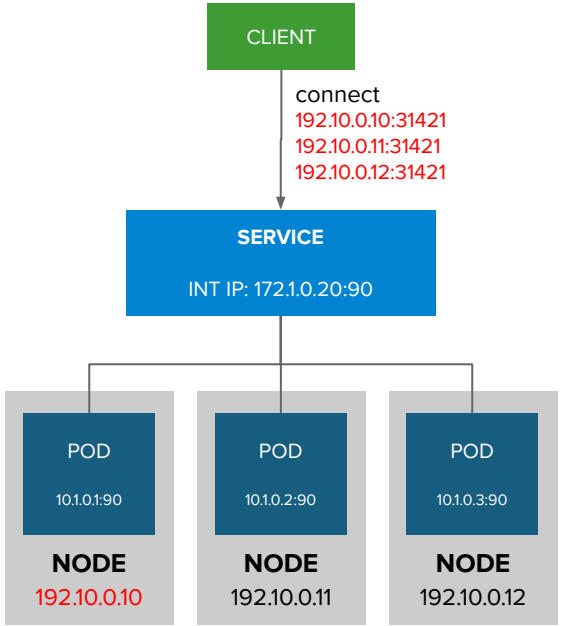


Alternative methods for ingress

Different ways that traffic can enter the cluster without the router

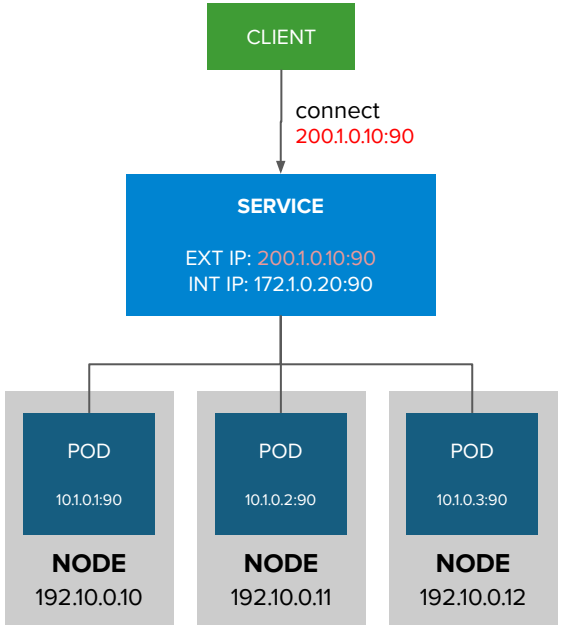
Entering the cluster on a random port with service nodeports

- NodePort binds a service to a unique port on all the nodes
- Traffic received on any node redirects to a node with the running service
- Ports in 30K-60K range which usually differs from the service
- Firewall rules must allow traffic to all nodes on the specific port



External traffic to a service on any port with external IP

- Access a service with an external IP on any TCP/UDP port, such as
  - Databases
  - Message Brokers
- Automatic IP allocation from a predefined pool using Ingress IP Self-Service
- IP failover pods provide high availability for the IP pool (fully supported in 4.8)



Cluster DNS

An automated system for providing hostname resolution within kubernetes

CoreDNS

- Built-in internal DNS to reach services by a (fully qualified) hostname
- Split DNS is used with CoreDNS
  - CoreDNS answers DNS queries for internal/cluster services
  - Other defined “upstream” name servers serve the rest of the queries

CNI ecosystem

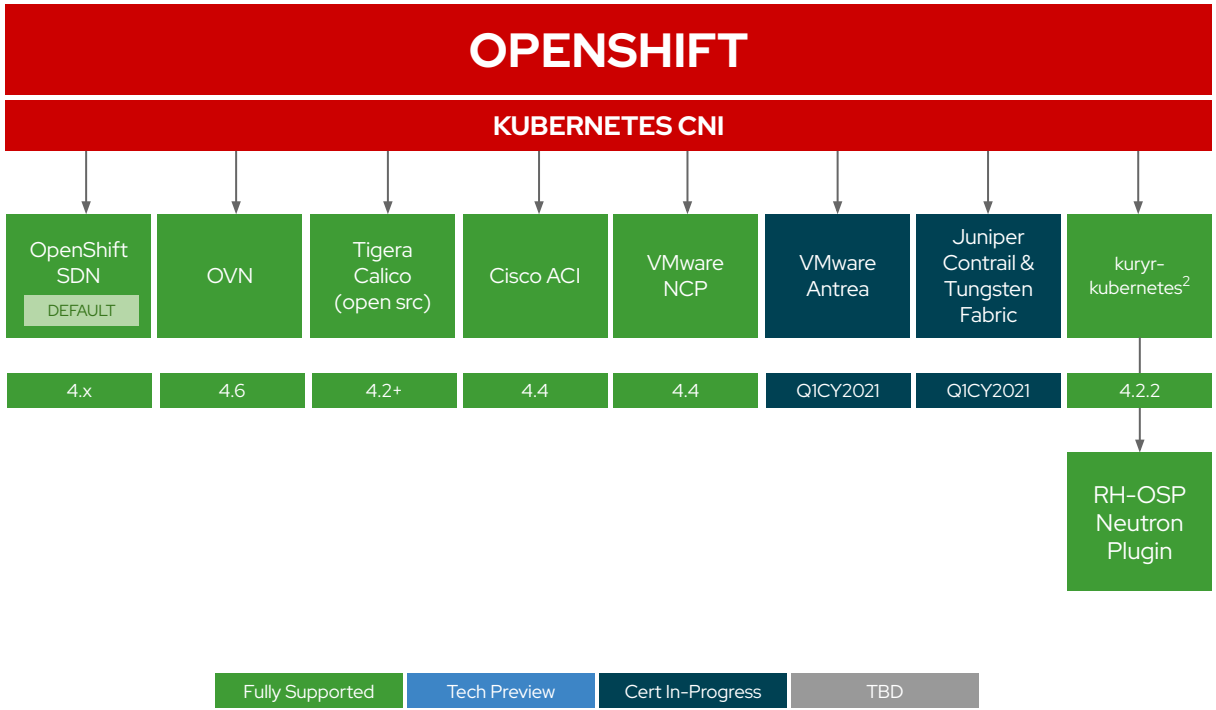
A pluggable model for network interface controls in kubernetes

Red Hat

OpenShift Networking Plug-ins

3rd-party Kubernetes CNI plug-in certification primarily consists of:

1. Formalizing the partnership
2. Certifying the container(s)
3. Certifying the Operator
4. Successfully passing the same Kubernetes networking conformance tests that OpenShift uses to validate its own SDN



OpenShift SDN

An Open

vSwitch-based

Software Defined

Network for

kubernetes

Red Hat

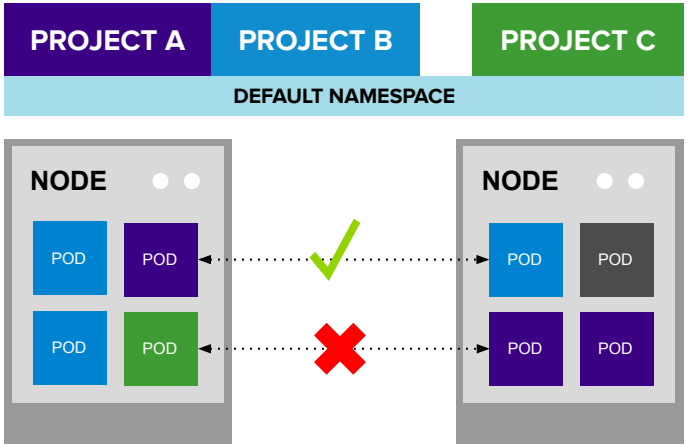
OpenShift SDN “flavors”

OPEN NETWORK **(Default)**

- All pods can communicate with each other across projects

MULTI-TENANT NETWORK

- Project-level network isolation
- Multicast support
- Egress network policies



Multi-Tenant Network

SDN (Re)configuration

Networking Advanced Settings

These are the OpenShift SDN settings that can be tweaked at install-time:

- Mode: NetworkPolicy, Multitenant, Subnet
- VXLAN Port Number
- MTU (autodetected, once)
- External OpenVSwitch

[How to Modify Advanced Network Configuration Parameters]

```
spec:
  defaultNetwork:
    type: OpenShiftSDN
    openshiftSDNConfig:
      mode: NetworkPolicy
      vxlanPort: 4789
      mtu: 1450
      useExternalOpenvswitch: false
```

NOTE: *Most network settings cannot be changed safely and affect the entire cluster.* The operator will prevent unsafe changes. If you need to force a change to a *non-production* cluster, see the operator README for the command, but a cluster re-install is likely to be the better choice.

kube-proxy Re-configuration

kube-proxy Advanced Settings

These are the kube-proxy settings that can be tweaked at install-time:

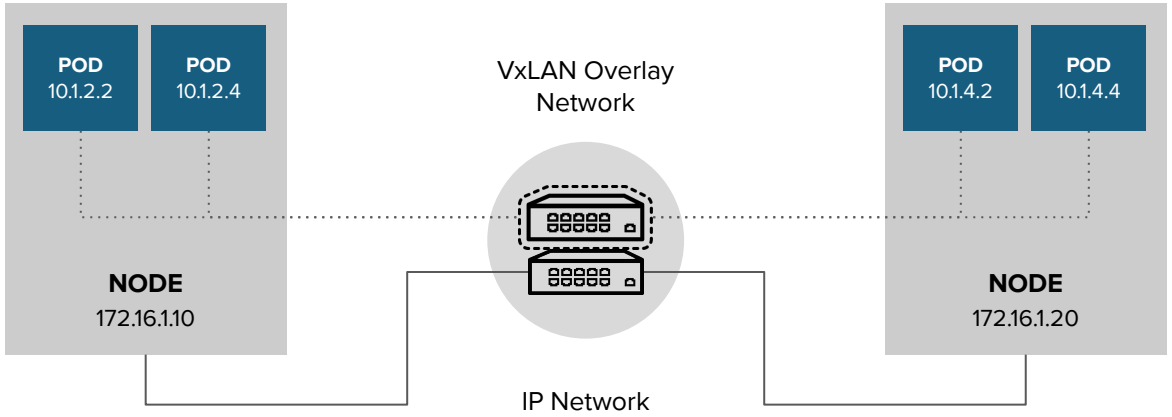
- iptablesSyncPeriod
- bindAddress
- proxyArguments (a list of kube-proxy command-line flags)

[How to Modify Advanced Network Configuration Parameters]

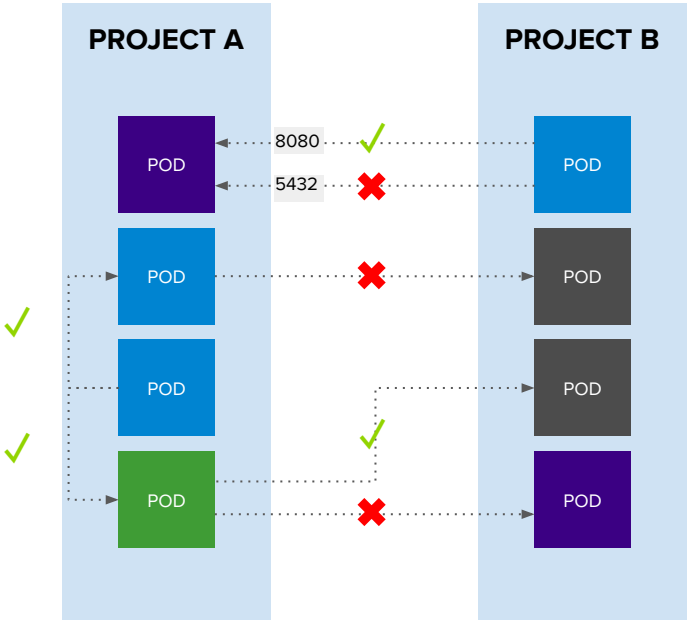
```
spec:
kubeProxyConfig:
   iptablesSyncPeriod: 30s
   bindAddress: 0.0.0.0
   proxyArguments:
     iptables-min-sync-period: ["30s"]
```

NOTE: *Most network settings cannot be changed safely and affect the entire cluster.* The operator will prevent unsafe changes. If you need to force a change to a *non-production* cluster, see the operator README for the command, but a cluster re-install is likely to be the better choice.

OpenShift SDN high-level architecture



NetworkPolicy



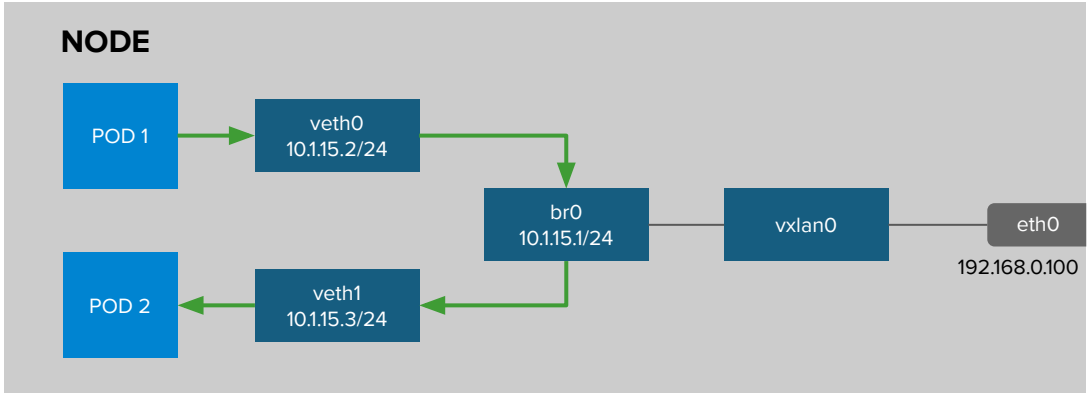
Example Policies

- Allow all traffic inside the project
- Allow traffic from green to gray
- Allow traffic to purple on 8080

```
apiVersion: extensions/v1beta1
kind: NetworkPolicy
metadata:
  name: allow-to-purple-on-8080
spec:
  podSelector:
    matchLabels:
      color: purple
  ingress:
  - ports:
    - protocol: tcp
      port: 8080
```

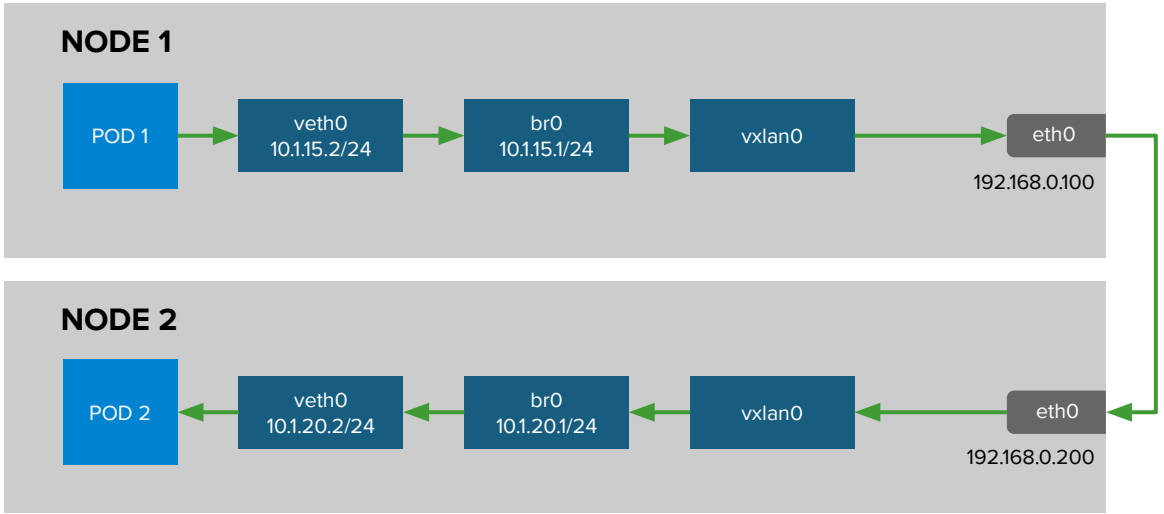
OpenShift SDN packet flows

container-container on same host



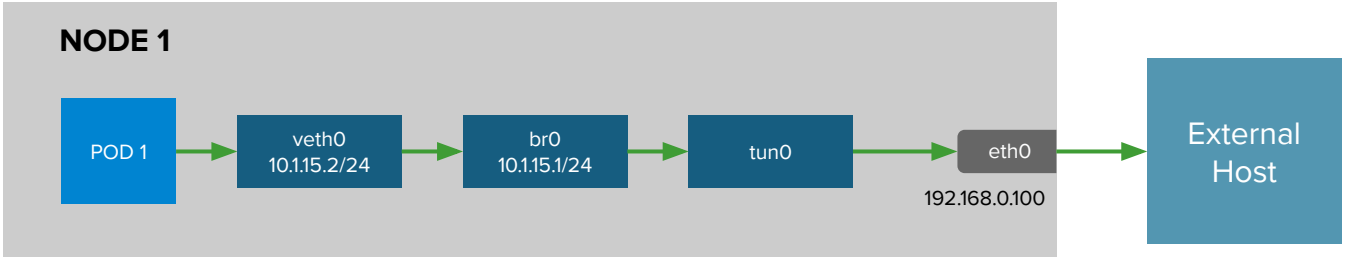
OpenShift SDN packet flows

container-container across hosts

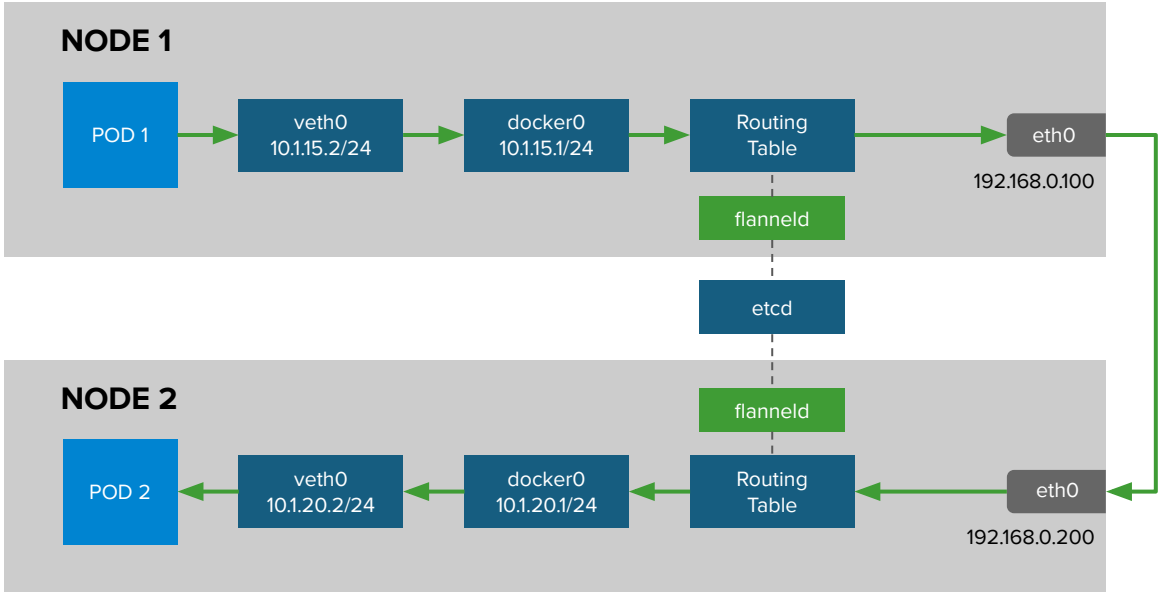


OpenShift SDN packet flows

container leaving the host



Kuryr and OpenStack



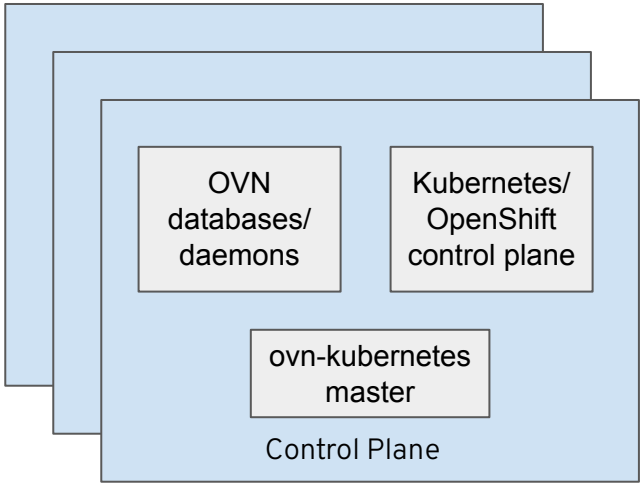
Flannel is minimally verified and is supported only and exactly as deployed in the OpenShift on OpenStack reference architecture https://access.redhat.com/articles/2743631

OVN

A Kubernetes-native networking solution

Red Hat

OVN Cluster Architecture

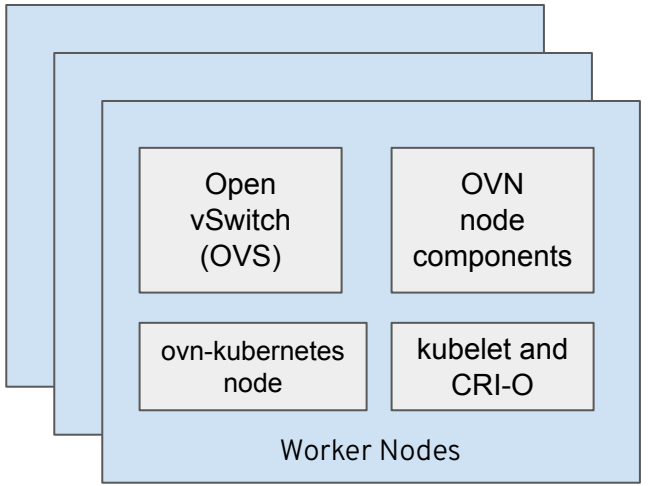


Project:

- OVN = Open Virtual Network
- OVN is a network virtualization platform based on Open vSwitch (OVS)
- Originally part of the OVS project, now a Linux Foundation project

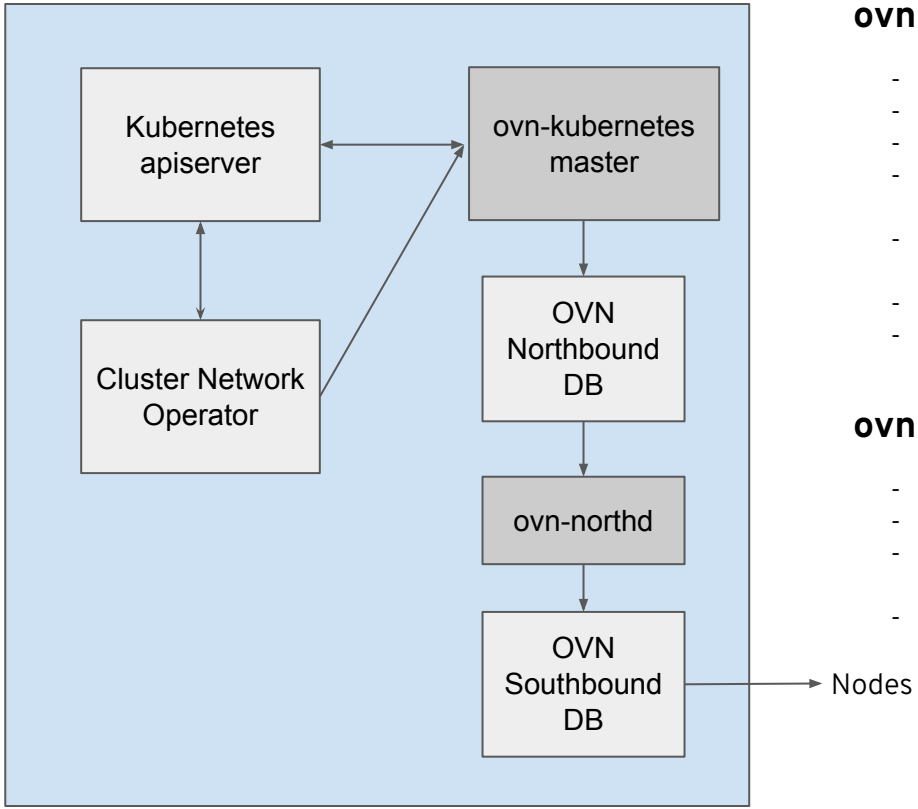
Features:

- Manages overlays and physical network connectivity
- Flexible security policies (ACLs)
- Distributed L3 routing, IPv4 and IPv6, L2/L3 Gateways
- Native support for NAT, load balancing, DHCP and RA
- Works with Linux, DPDK, and Hyper-V



Red Hat

Control Plane Architecture



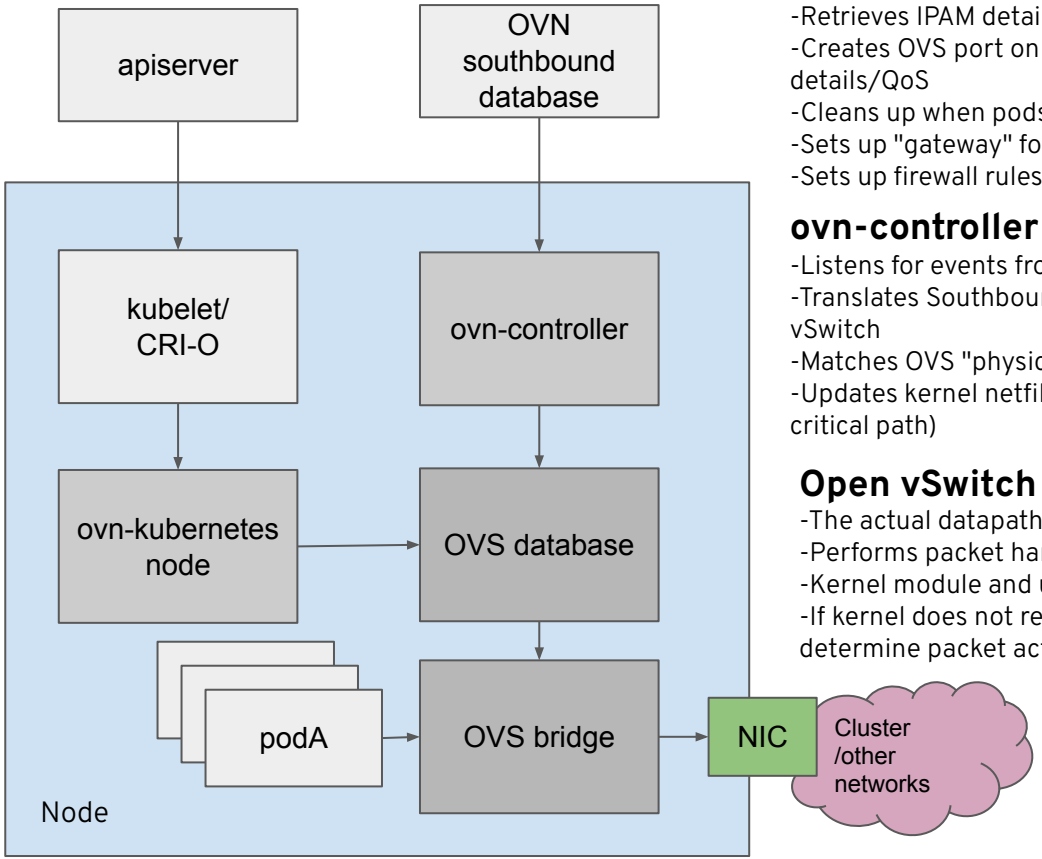
ovn-kubernetes master

- Deployment created by Cluster Network Operator
- Multiple instances elect a leader
- Kubernetes API is the single source of truth
- Listens for cluster events (Pods, Namespaces, Services, Endpoints, NetworkPolicy)
- Translates cluster events to OVN logical network elements (logical switches, ACLs, logical routers, switch ports)
- Handles all required IPAM
- Updates OVN Northbound database based on Kube API state

ovn-northd

- Multiple instances managed by ovn-kubernetes master
- Listens for Northbound DB changes
- Decomposes logical network elements into OVN pipeline and populates the Southbound DB
- Keeps no state itself (eg "cattle")

Worker Node Architecture



ovn-kubernetes node

-Called as a CNI plugin from kubelet/CRI-O runtime
-Retrieves IPAM details from Kube API (written by ovn-kubernetes master)
-Creates OVS port on bridge, moves it into pod network namespace, sets IP details/QoS
-Cleans up when pods die
-Sets up "gateway" for cluster-external network access
-Sets up firewall rules and routes for HostPort and Service access from node

ovn-controller

-Listens for events from OVN Southbound database
-Translates Southbound database into OpenFlow and programs local OVS vSwitch
-Matches OVS "physical" ports with OVN logical ports
-Updates kernel netfilter tables for load balancing functionality (no iptables in critical path)

Open vSwitch

-The actual datapath for containers
-Performs packet handling based on OpenFlow rules
-Kernel module and userspace management daemon
-If kernel does not recognize a flow, calls up to management daemon to determine packet action, then caches the match+action for later

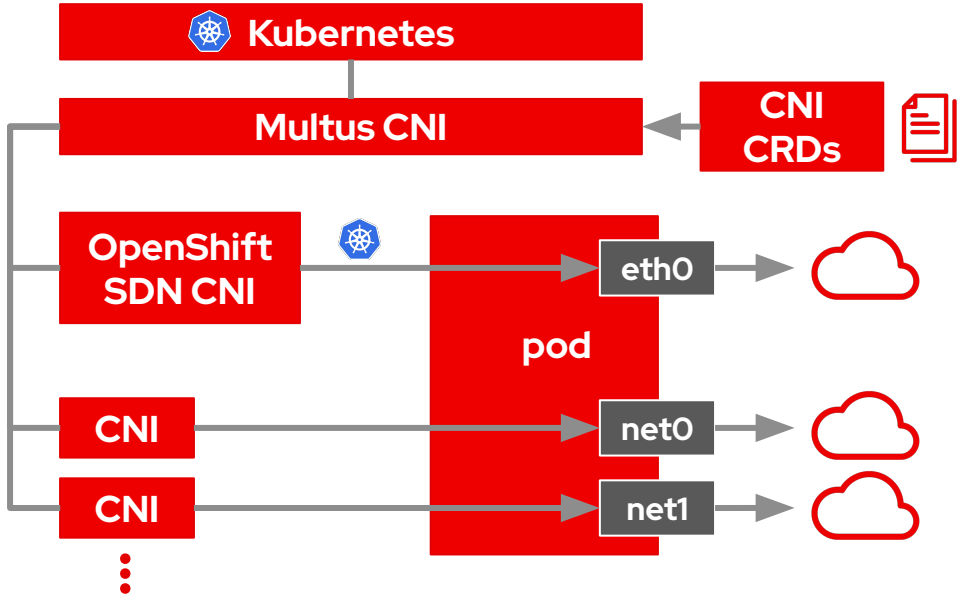
Multus

A CNI plugin that provides multiple network interfaces for pods

Multinetwork with Multus

The Multus CNI “meta plugin” for Kubernetes enables one to create multiple network interfaces per pod, and assign a CNI plugin to each interface created.

Additional networks and capability

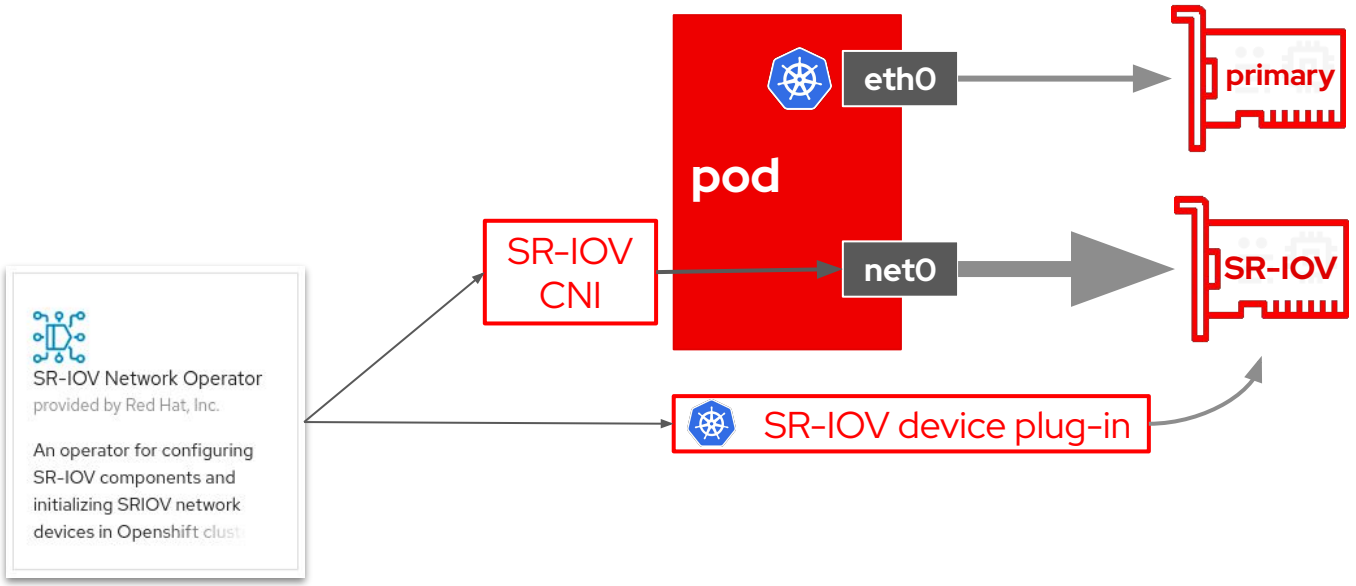


Additional OpenShift-Supported Secondary CNI Plug-Ins

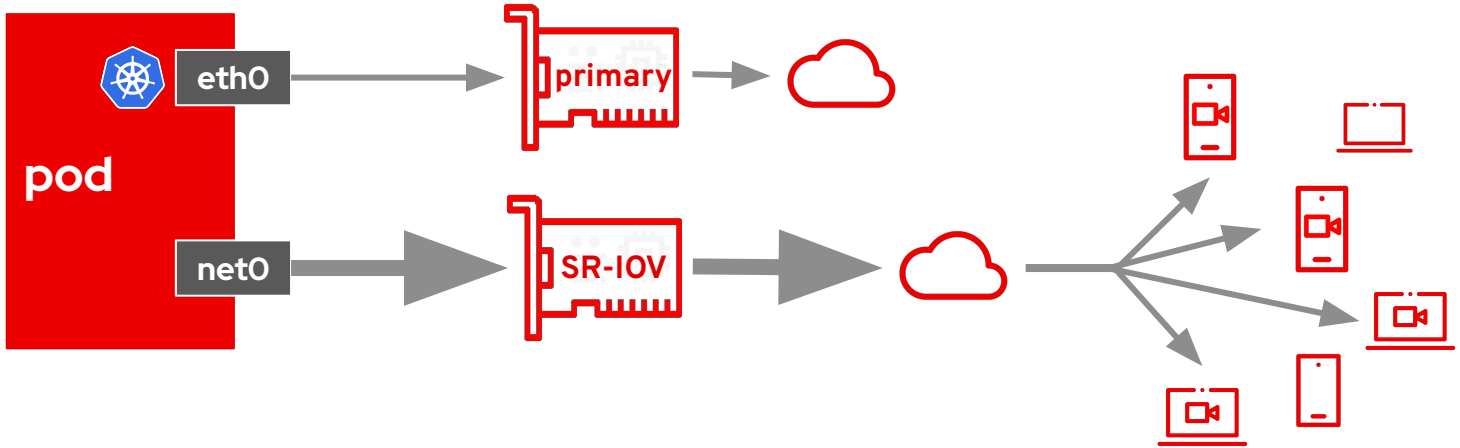
OpenShift 4.x Tested Integrations: Network Components and Plugins

- host device
- IPAM(dhcp)
- MACVLAN
- IPVLAN
- Bridge with VLAN
- Static IPAM
- DHCP IPAM
- Route Override
- whereabouts
- SR-IOV
- ...

SR-IOV



High-performance multicast



OpenShift Monitoring

An integrated cluster monitoring and alerting stack

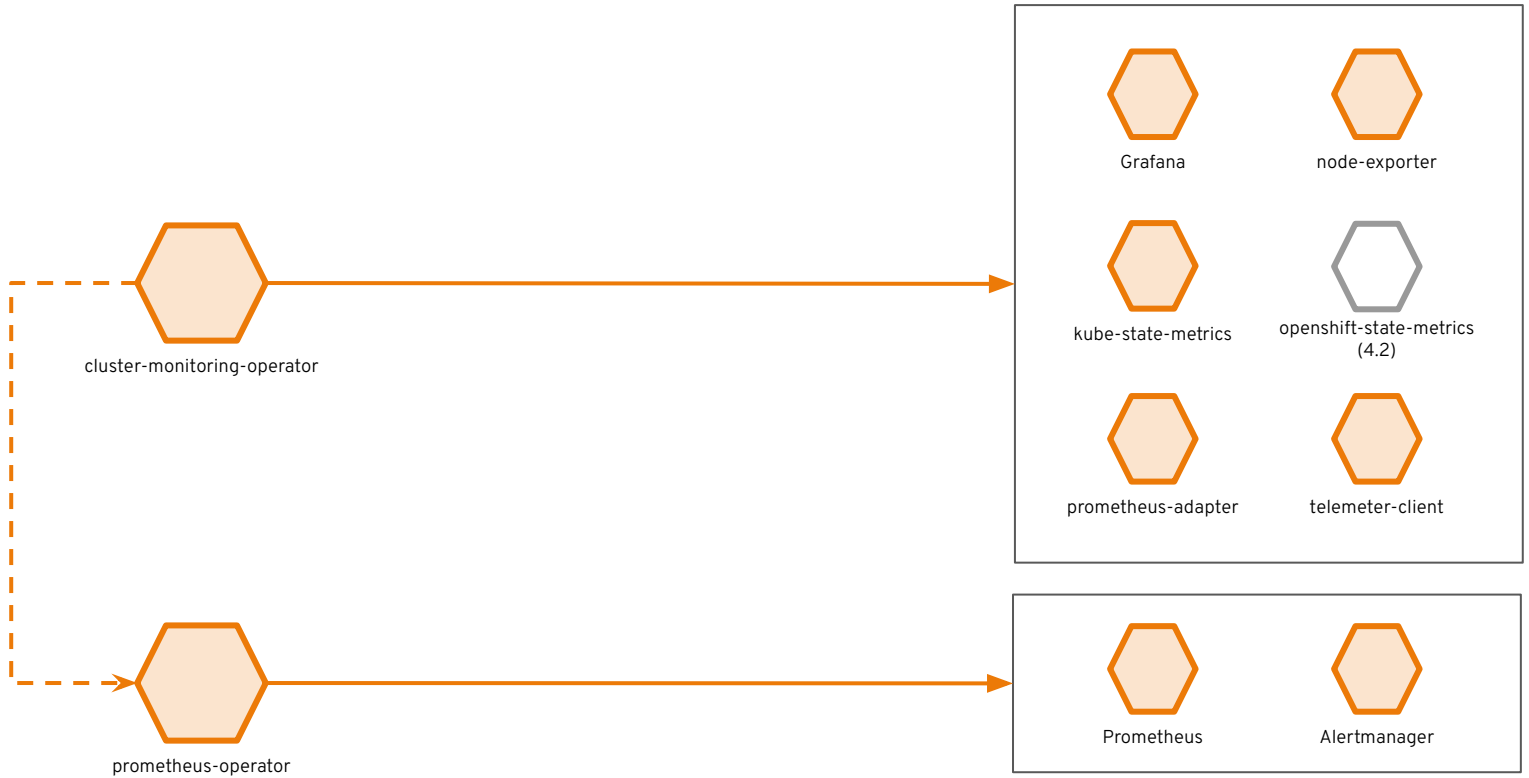
Red Hat

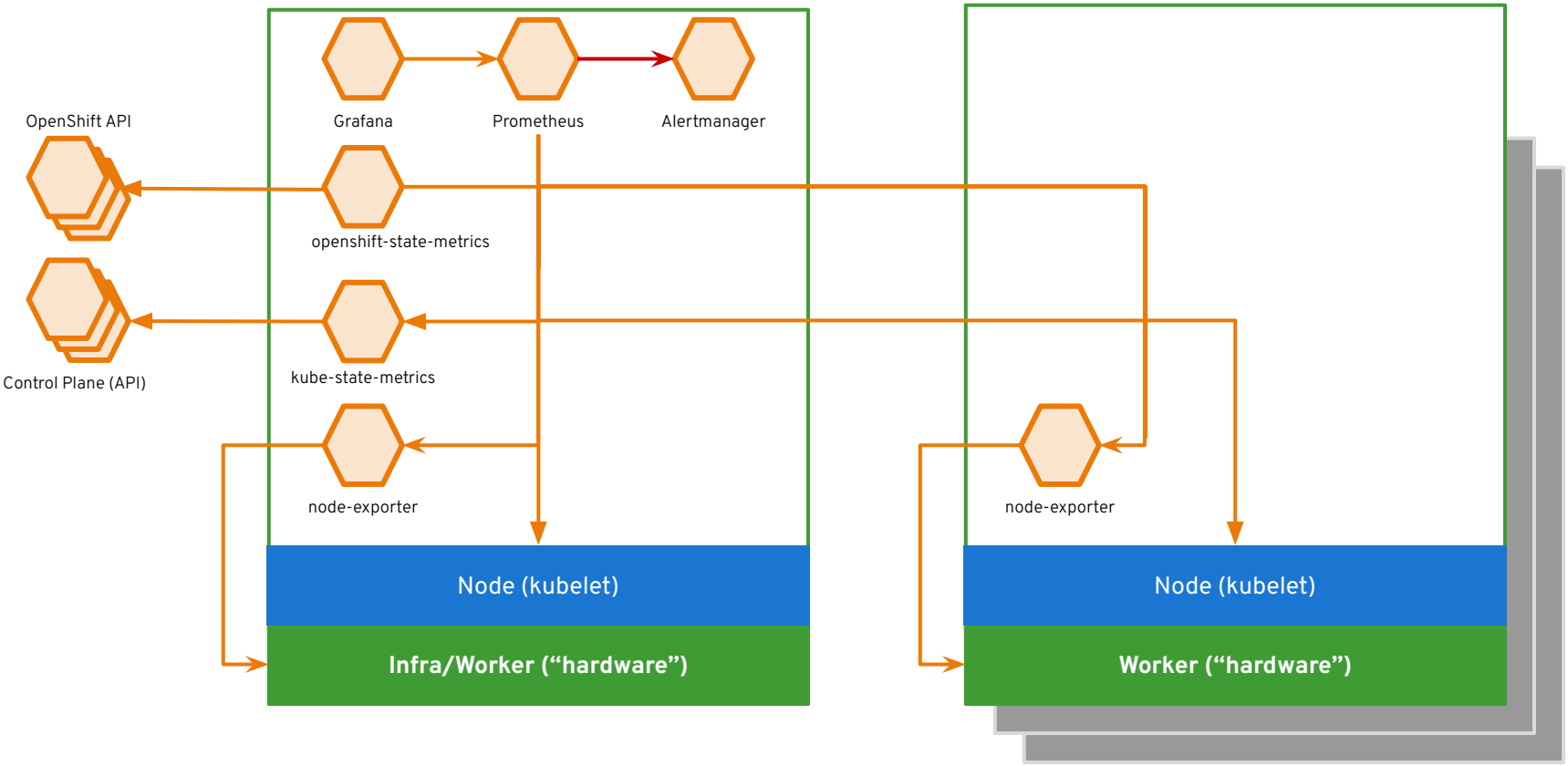
OpenShift Cluster Monitoring

Metrics collection and storage via Prometheus, an open-source monitoring system time series database.

Alerting/notification via Prometheus' Alertmanager, an open-source tool that handles alerts send by Prometheus.

Metrics visualization via Grafana, the leading metrics visualization technology.


Grafana
node-exporter
kube-state-metrics
openshift-state-metrics
(4.2)
prometheus-adapter
telemeter-client
cluster-monitoring-operator
prometheus-operator
Prometheus
Alertmanager


OpenShift API
Grafana
Prometheus
Alertmanager
openshift-state-metrics
Control Plane (API)
kube-state-metrics
node-exporter
node-exporter
Node (kubelet)
Infra/Worker (“hardware”)
Node (kubelet)
Worker (“hardware”)

OpenShift Logging

An integrated solution for exploring and corroborating application logs

Red Hat

Observability via log exploration and corroboration with EFK

Components

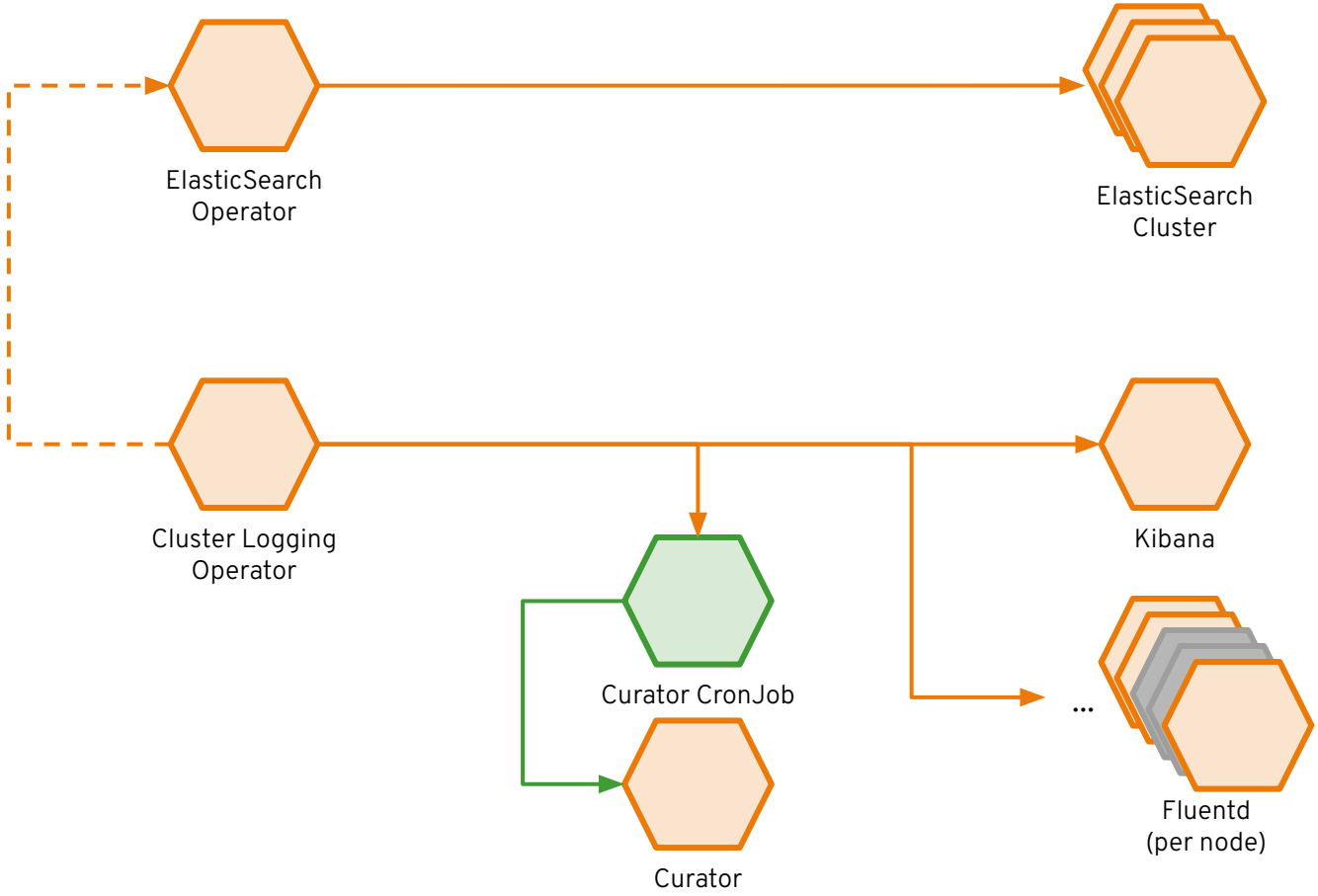
- **Elasticsearch:** a search and analytics engine to store logs
- **Fluentd:** gathers logs and sends to Elasticsearch.
- **Kibana:** A web UI for Elasticsearch.

Access control

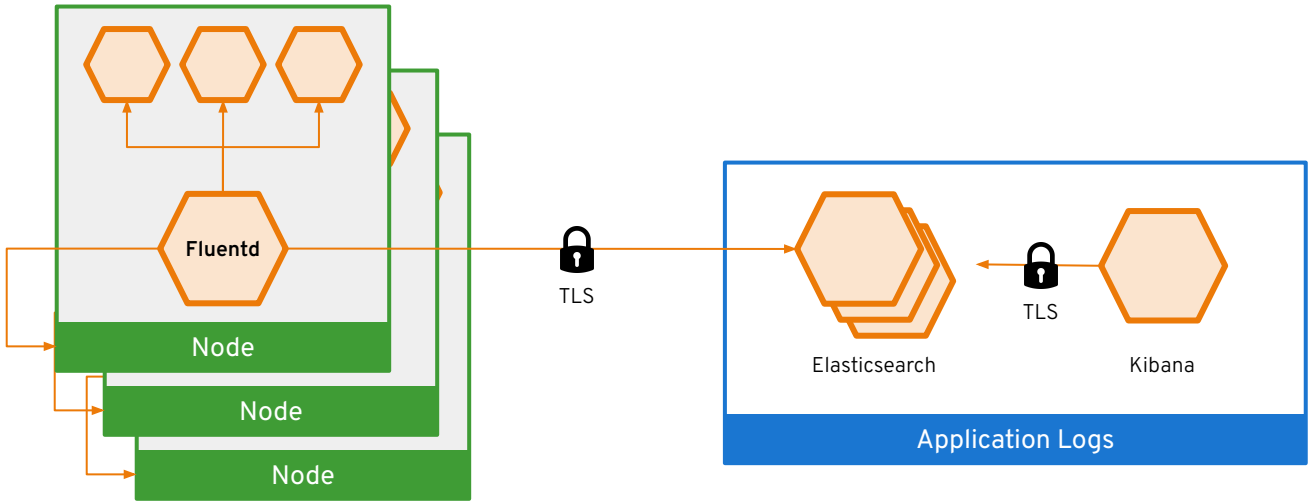
- Cluster administrators can view all logs
- Users can only view logs for their projects

Ability to forward logs elsewhere

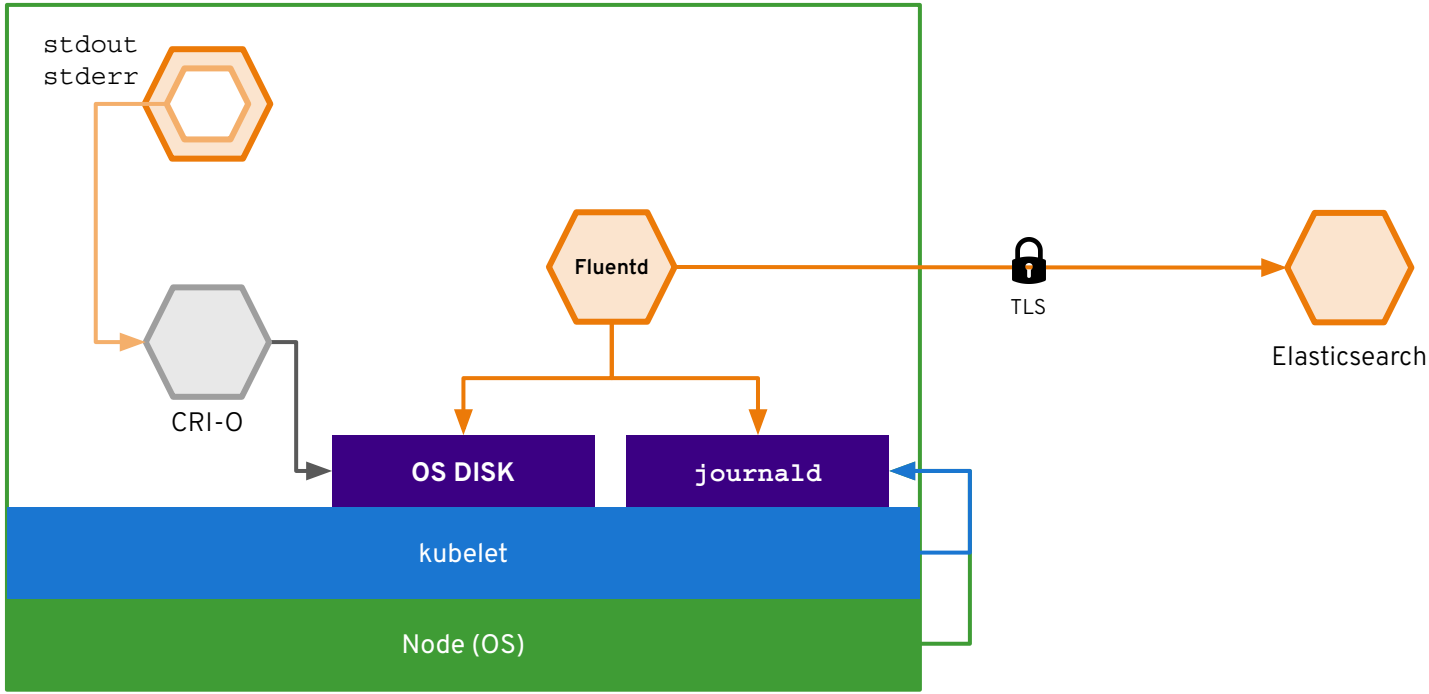
- External elasticsearch, Splunk, etc


ElasticSearch
Operator
ElasticSearch
Cluster
Cluster Logging
Operator
Kibana
Curator CronJob
...
Fluentd
(per node)
Curator

Log data flow in OpenShift



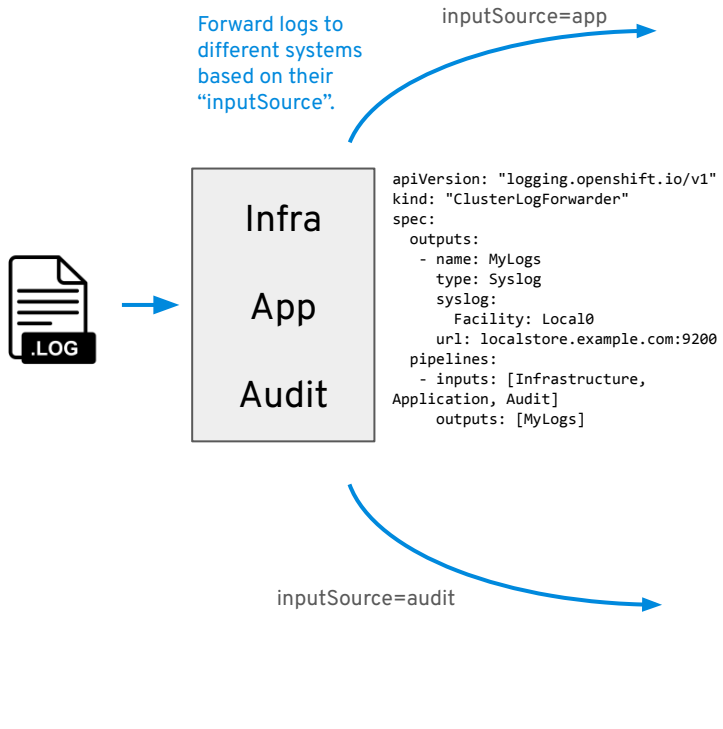
Log data flow in OpenShift



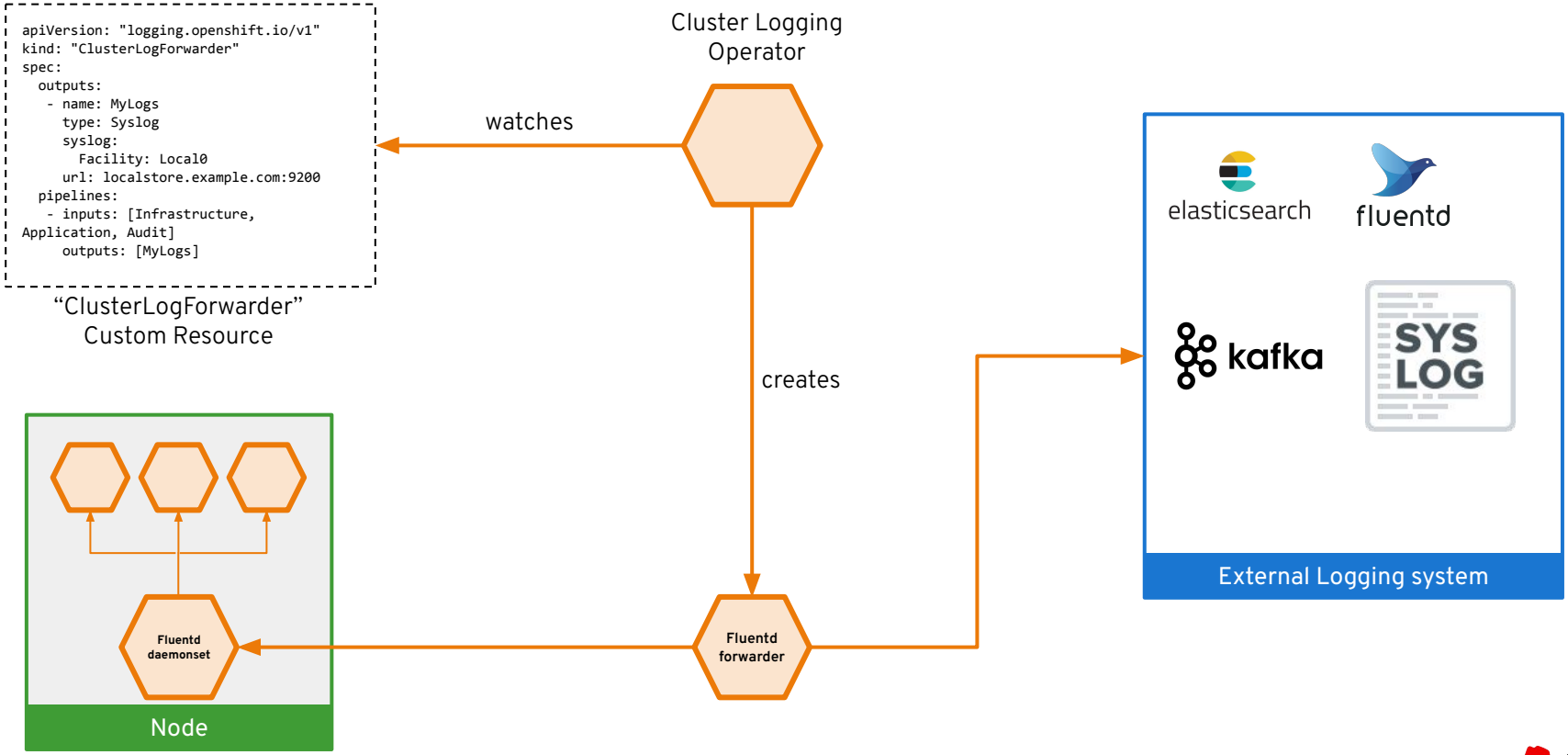
New log forwarding API (since 4.6)

Abstract Fluentd configuration by introducing new log forwarding API to improve support and experience for customers.

- Introducing a new, cluster-wide *ClusterLogForwarder* CRD (API) that replaces needs to configure log forwarding via Fluentd ConfigMap.
- The API helps to reduce probability to misconfigure Fluentd and helps bringing in more stability into the Logging stack.
- Features include: Audit log collection and forwarding, Kafka support, namespace- and source-based routing, tagging, as well as improvements to the existing log forwarding features (e.g. syslog RFC5424 support).



Secure Log Forwarding to 3rd party



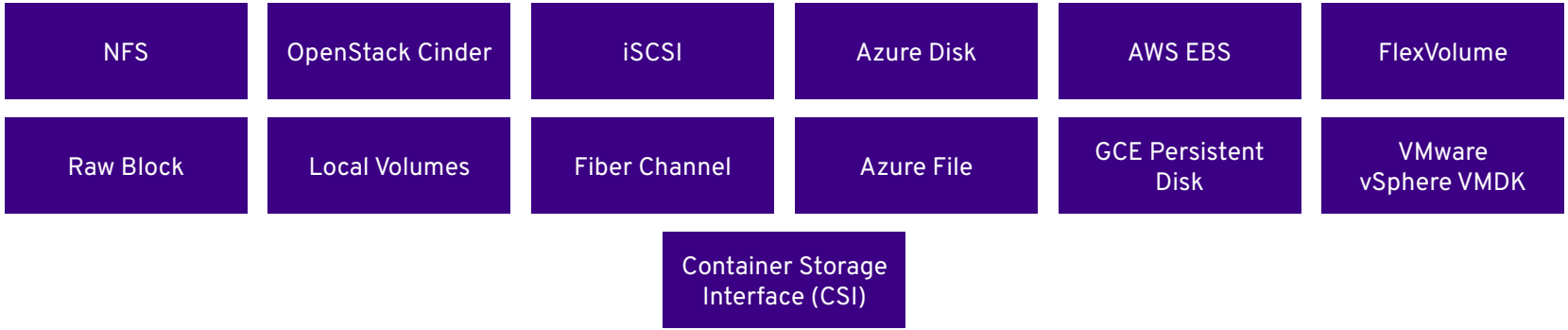
Red Hat

Persistent Storage

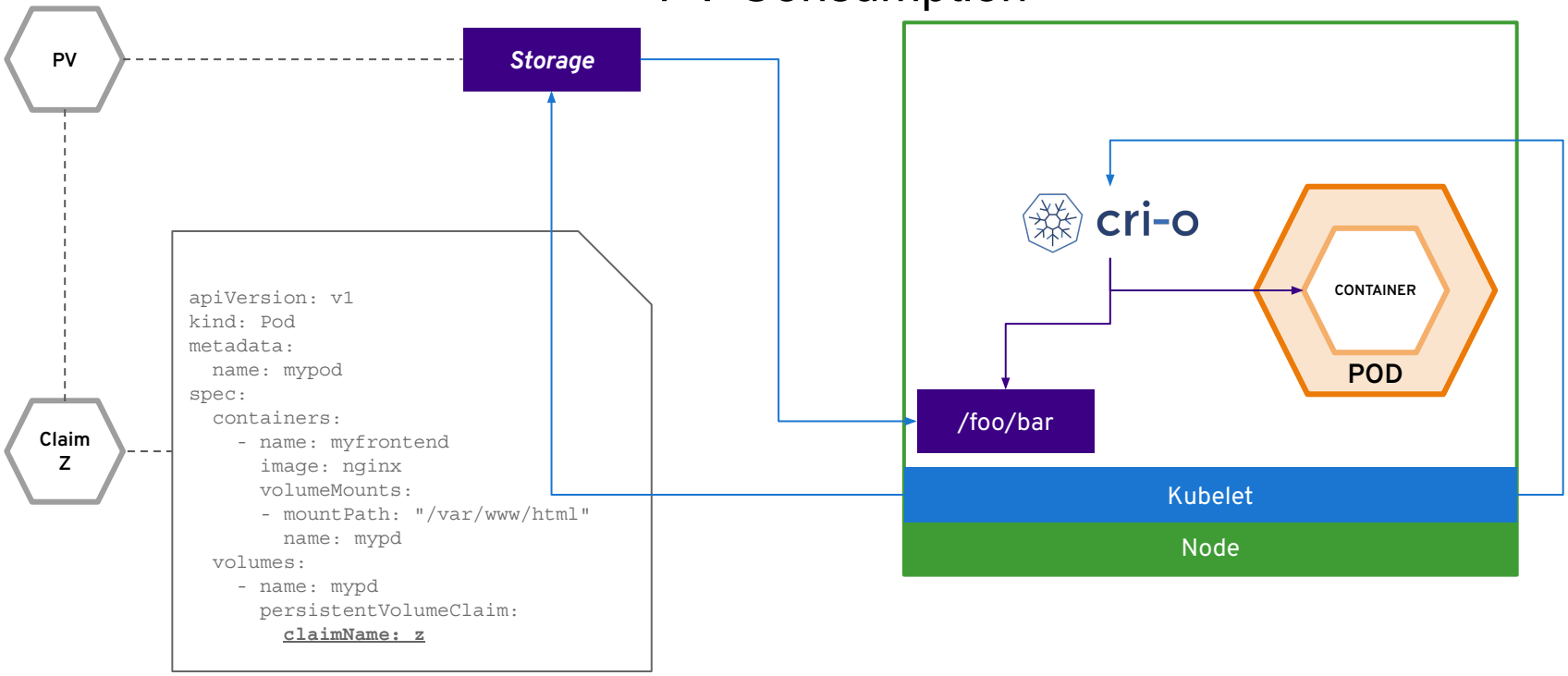
Connecting real-world storage to your containers to enable stateful applications

Red Hat

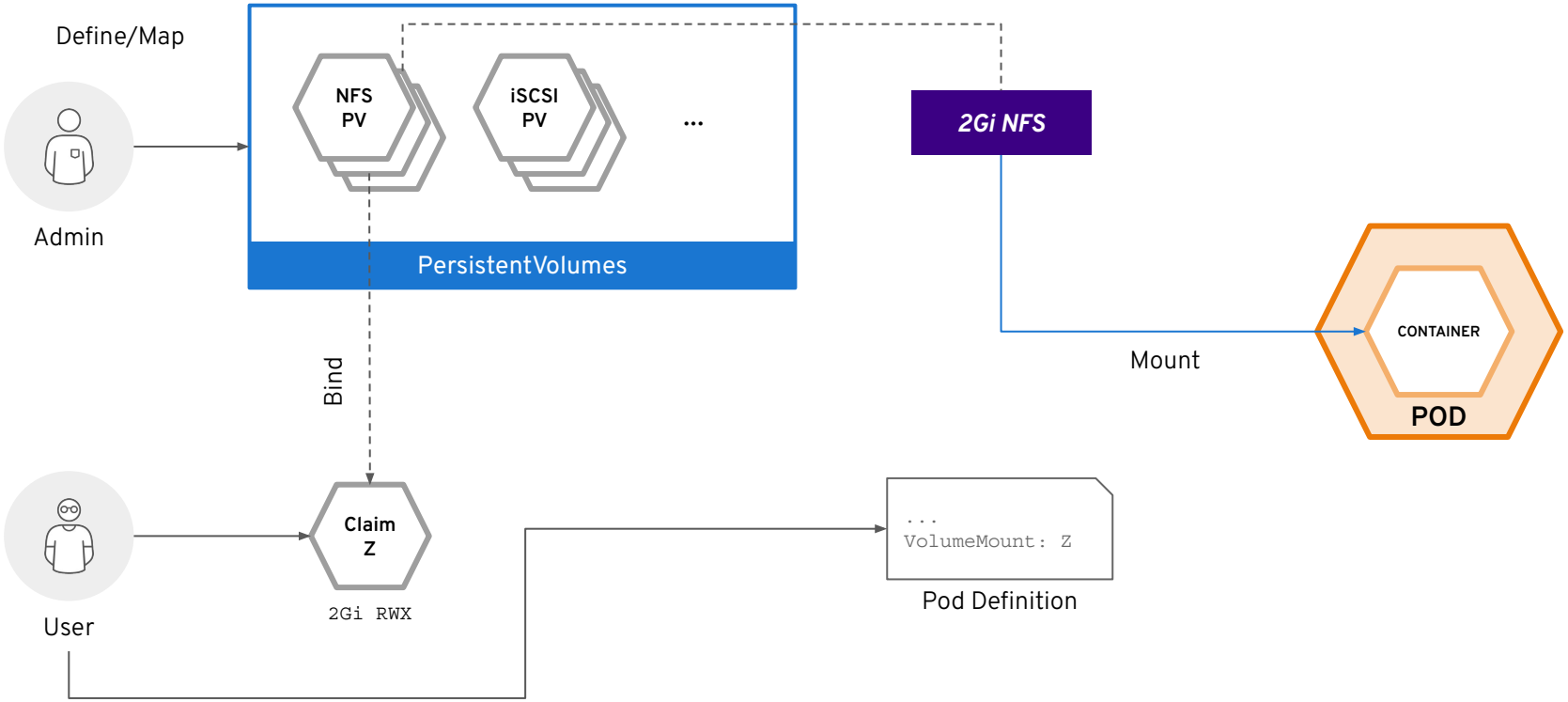
A broad spectrum of static and dynamic storage endpoints



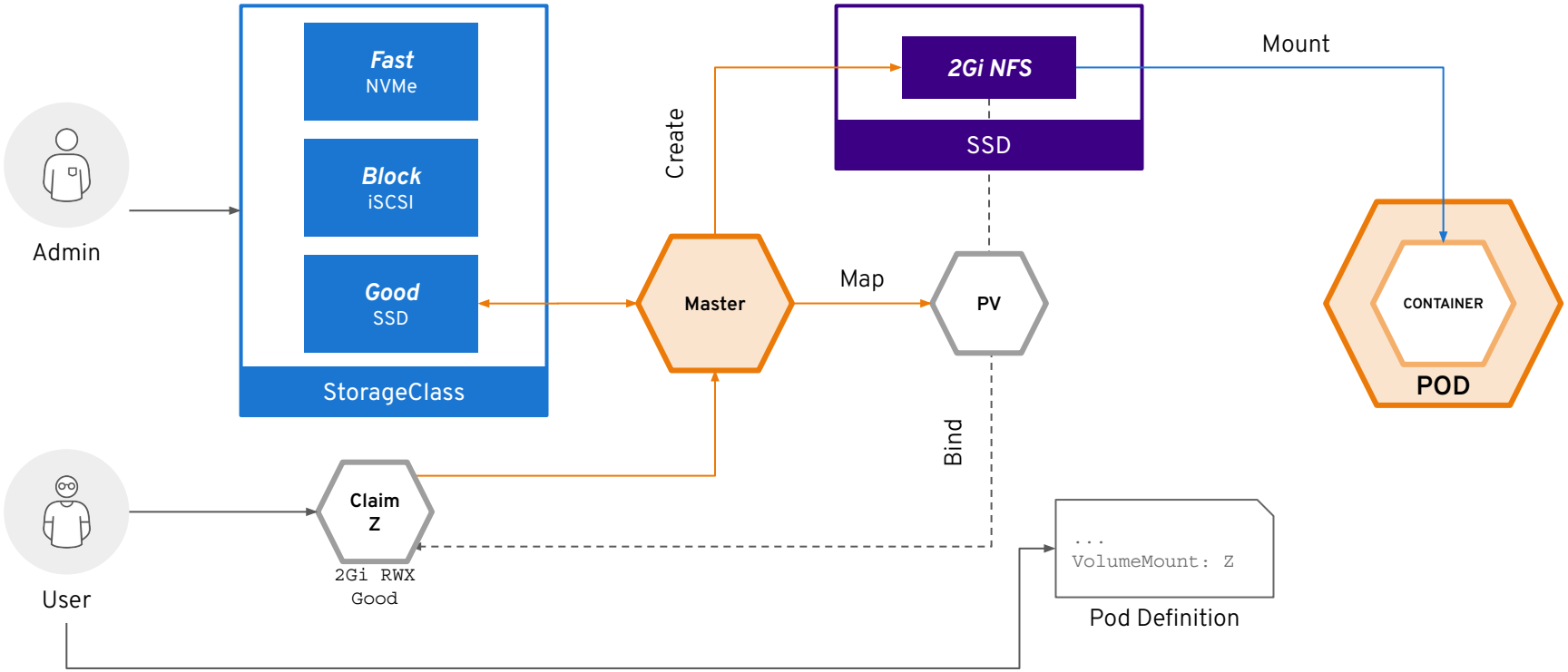
PV Consumption



Static Storage Provisioning

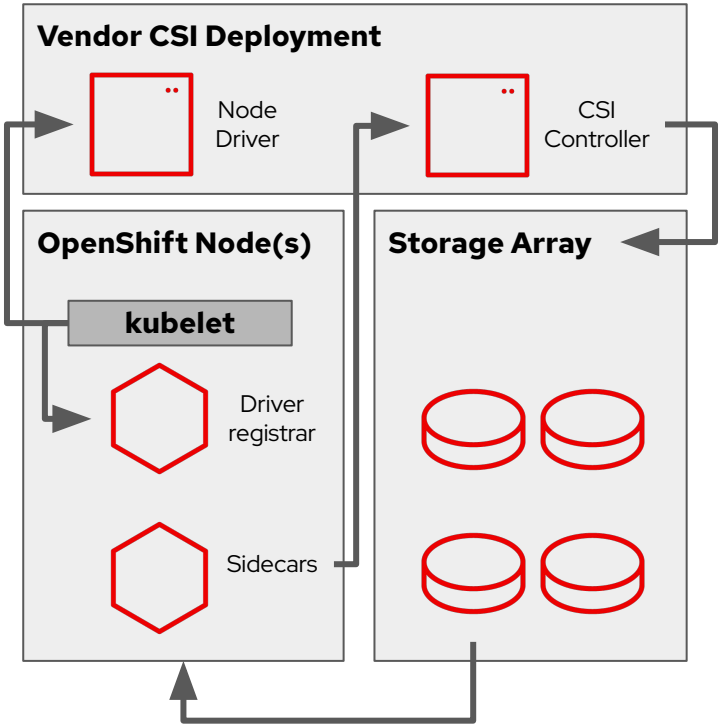


Dynamic Storage Provisioning



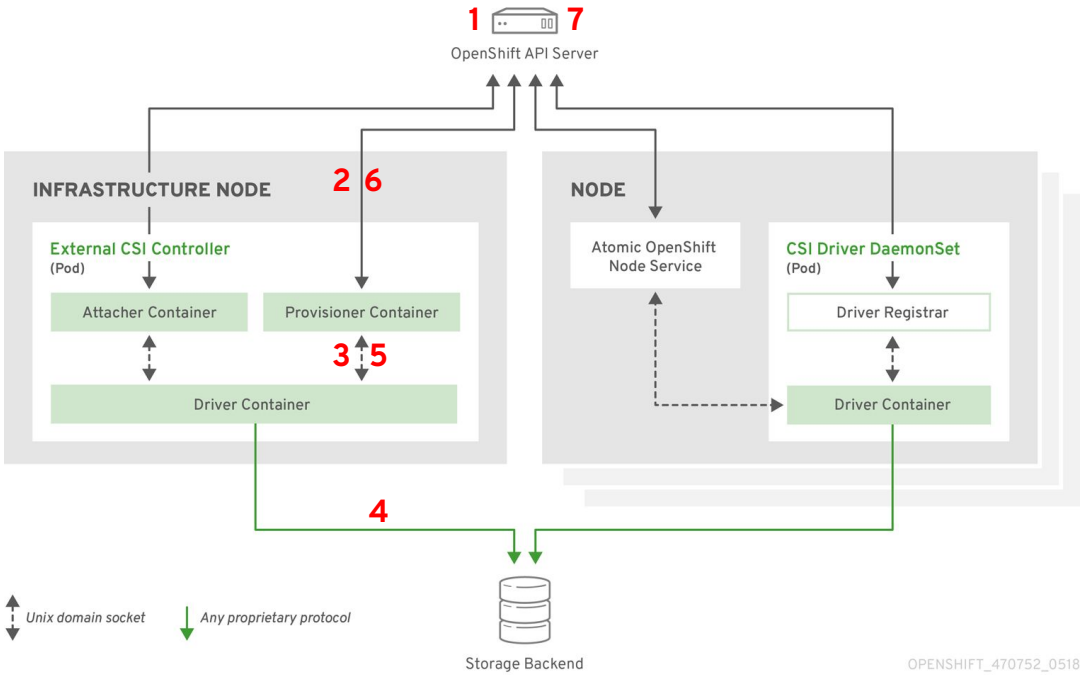
CSI Driver Paradigm

- CSI drivers and logic are provided by storage vendors
  - Each implementation may be different based on the vendor
- Controller logic is deployed to the OpenShift cluster as an Operator, deployment, or even a standalone Pod(s)
  - Responsible for interfacing with storage device to create and manage volumes, snapshots, clones, etc.
  - Respond to events (create, delete PVC) for assigned StorageClass(es)
  - Sidecars assist with hooks for additional functionality - snapshots, resizing, etc.
- Each node hosts, via a DaemonSet, one or more CSI node plugin Pods for the driver
  - Kubelet requests the node plugin to mount/unmount volumes, format block devices if needed, etc.



CSI Dynamic Provisioning

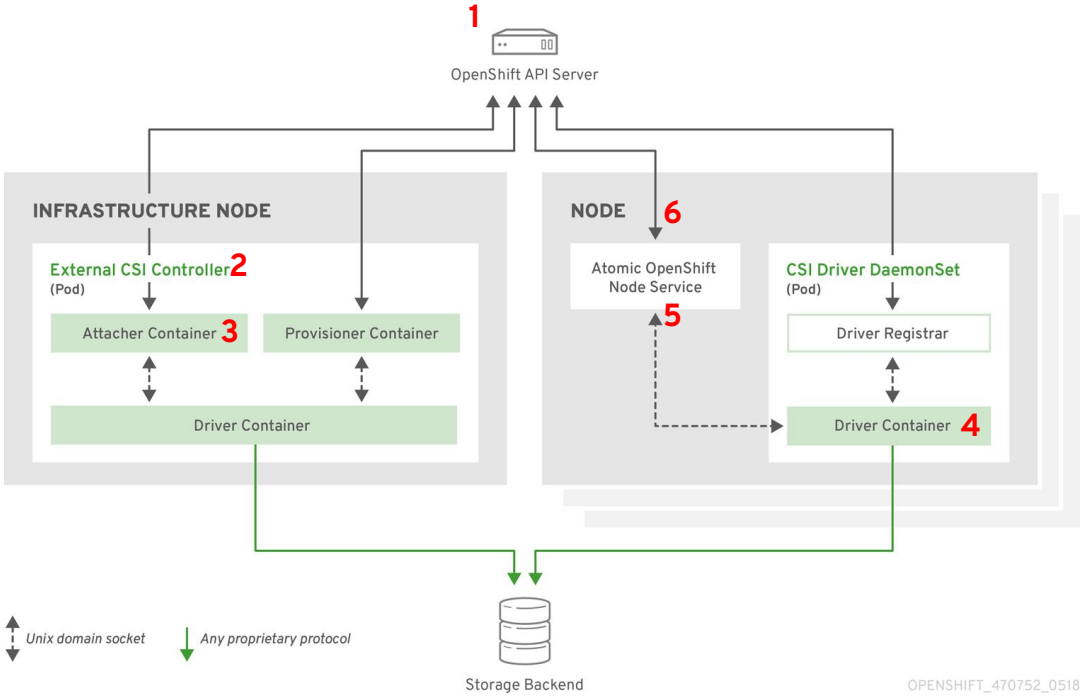
1. User creates a PVC
2. The external provisioner gets an event that a new PVC was created
3. The external provisioner initiates CreateVolume call to the CSI driver
4. The CSI driver talks to storage backend and creates a volume
5. The CSI driver returns a volume to the external provisioner
6. The external provisioner creates PV on API server
7. Kubernetes PV controller finishes the binding (PVC is Bound)



Red Hat

CSI Volume Mounting

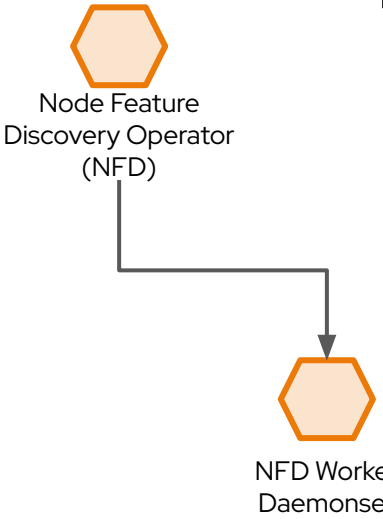
1. User instantiates a Pod with a PVC
2. The CSI controller is notified of a volume publish event via the attacher sidecar
3. The CSI controller takes any actions on the storage device to make the volume mountable, e.g. NFS export rules
4. The node driver stages the volume, taking action to prepare the volume to be used, e.g. formatting a non-raw block device
5. The node driver mounts the volume at the location requested by Kubelet
6. The volume is attached to the container, by Kubelet, as defined

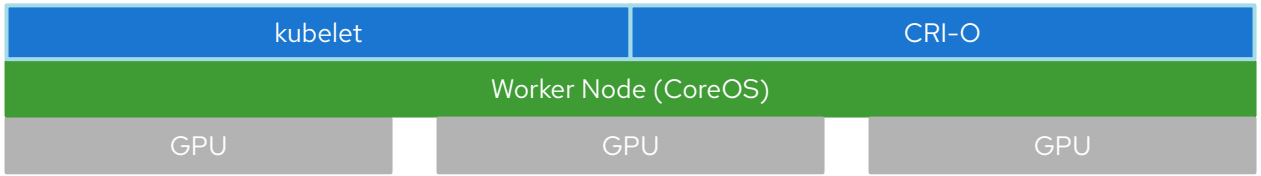


Special Resources and Devices

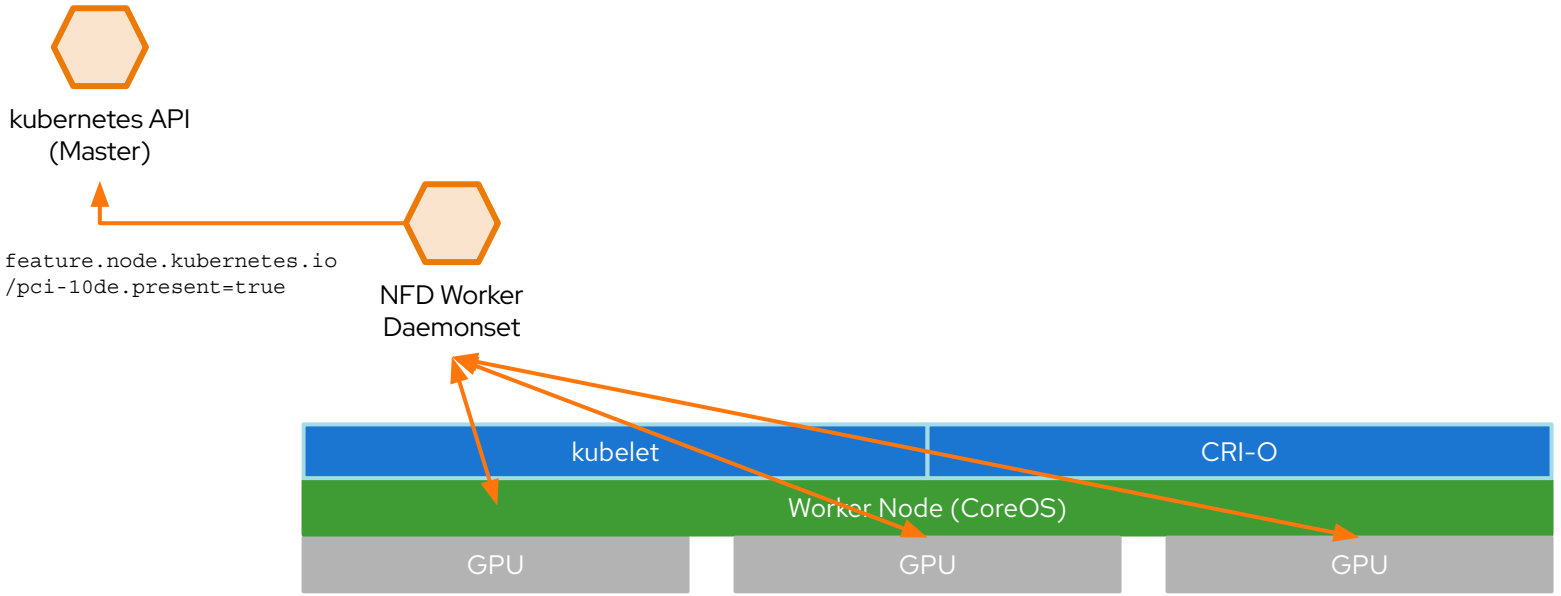
Enabling GPU, network, and other specialty resources for workloads

NFD finds certain resources

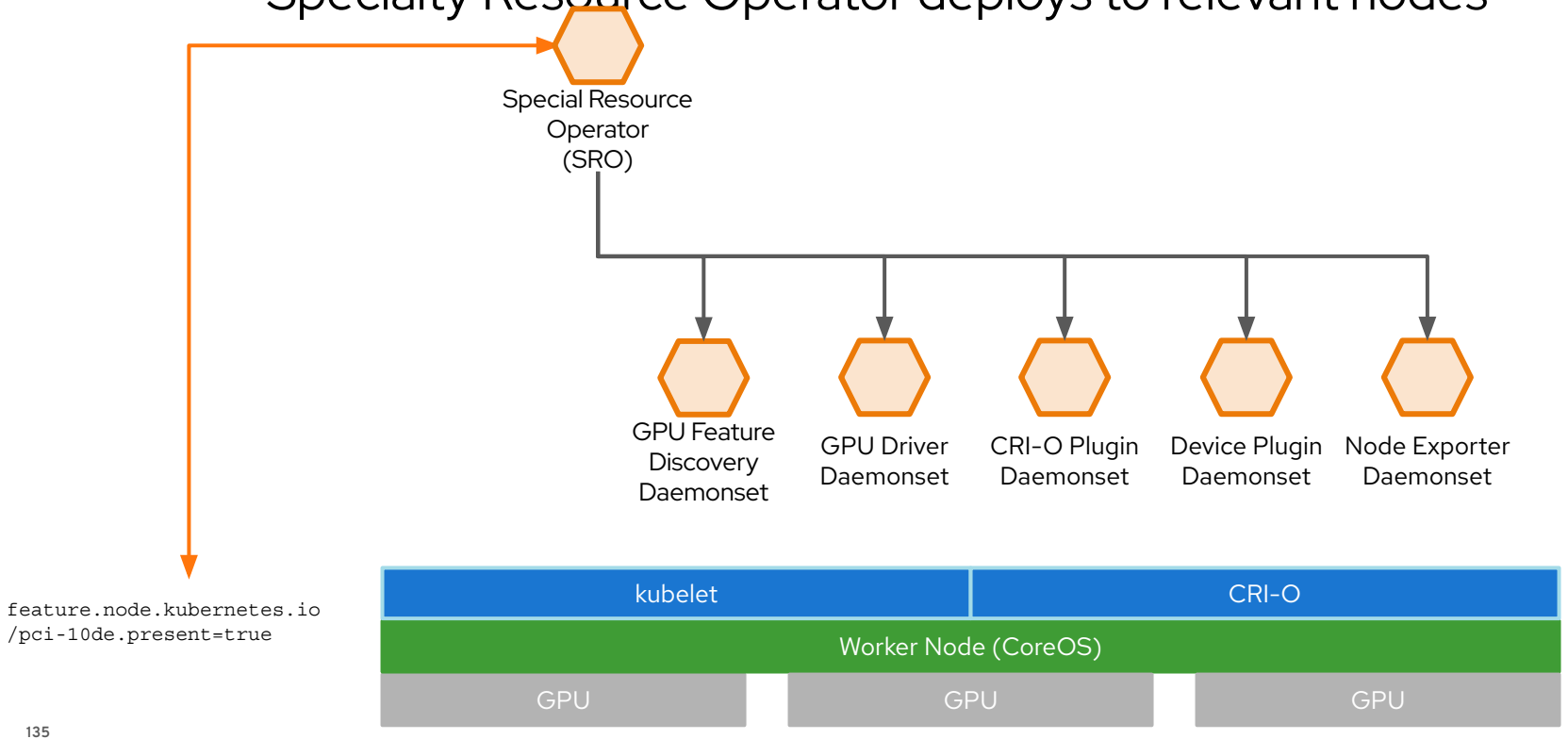




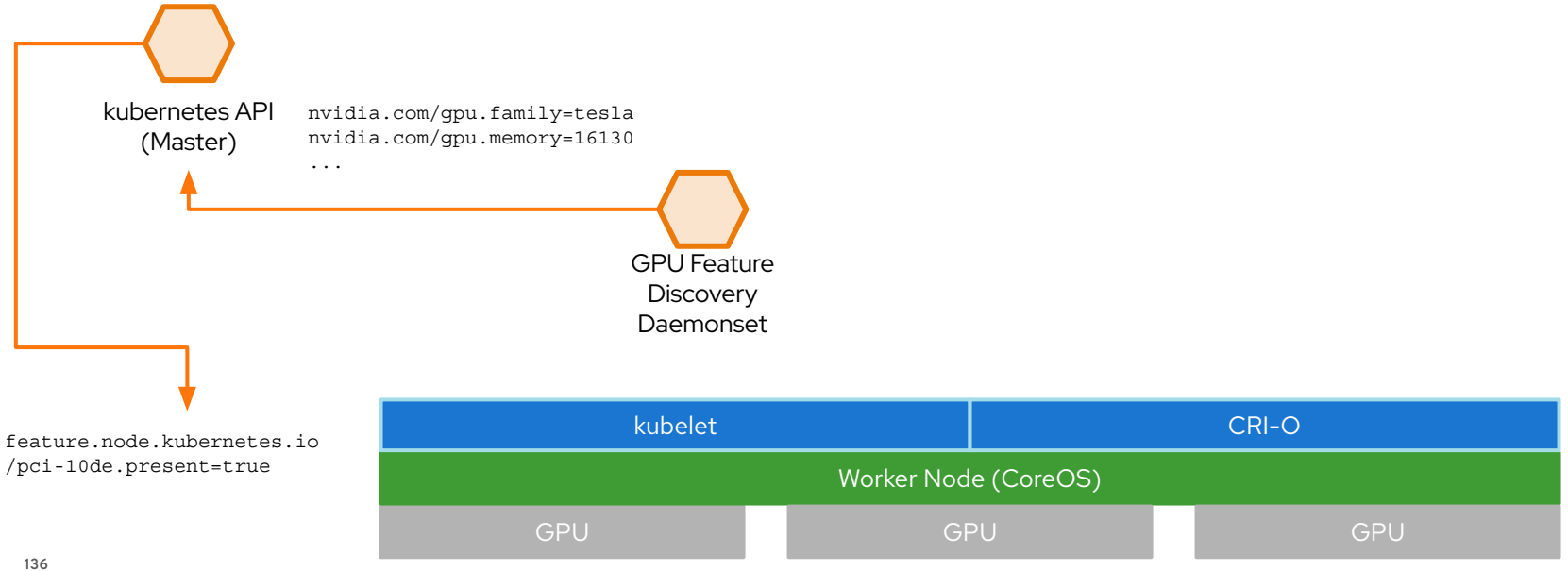
NFD labels nodes



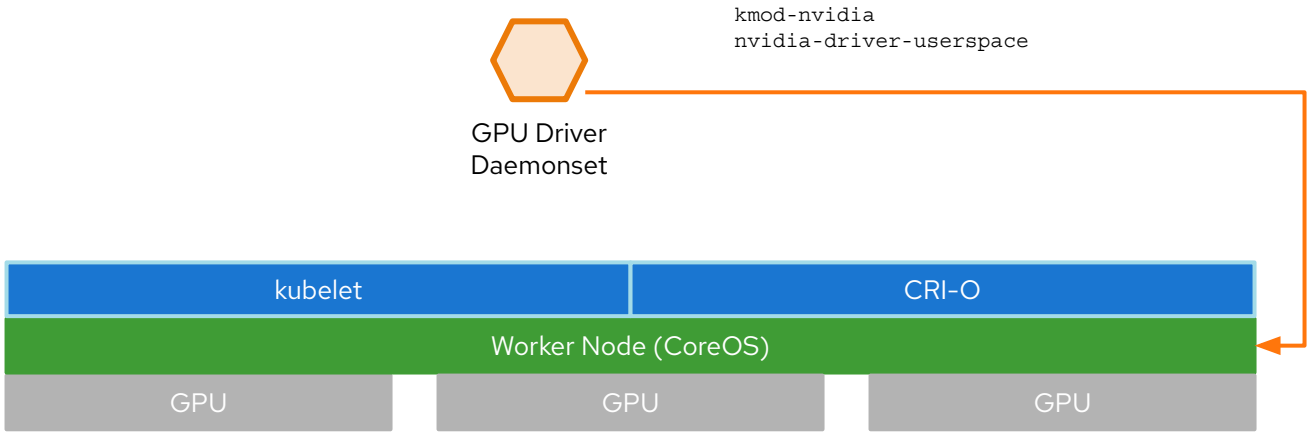
Specialty Resource Operator deploys to relevant nodes



GPU Feature Discovery reports additional capabilities

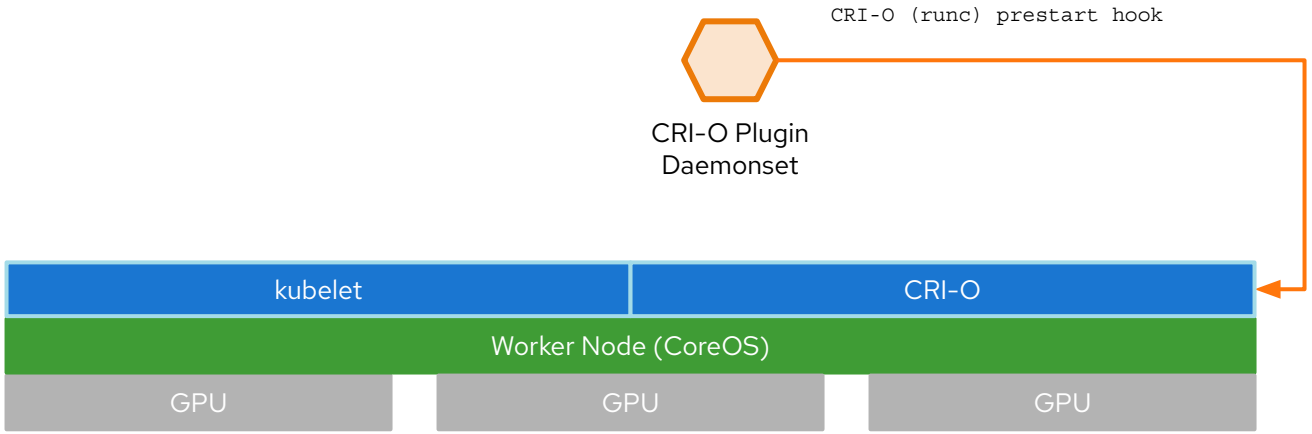


GPU Driver installs kmod and userspace drivers



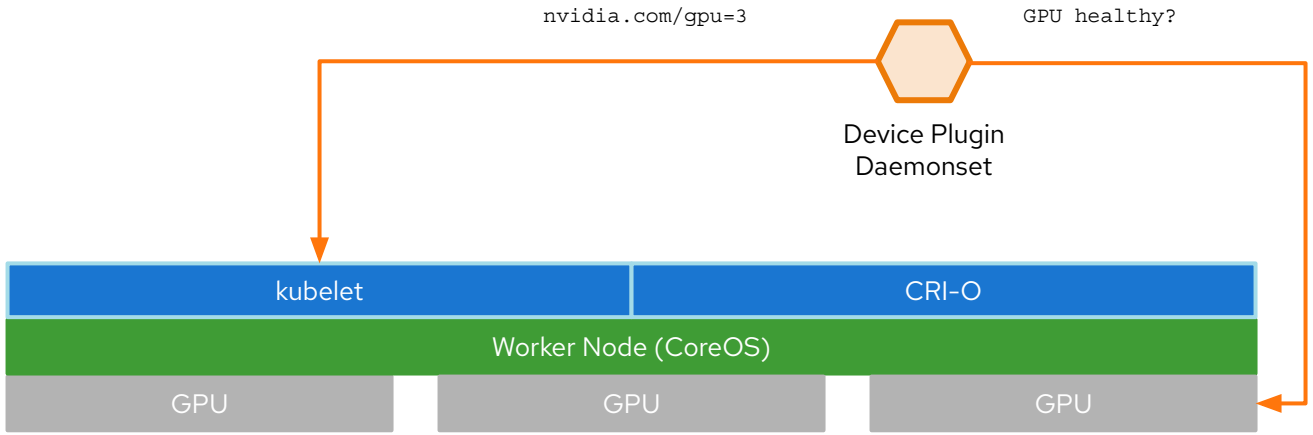

```
feature.node.kubernetes.io
/pci-10de.present=true
nvidia.com/gpu.family=tesla
nvidia.com/gpu.memory=16130
...
```

CRI-O Plugin installs prestart hook



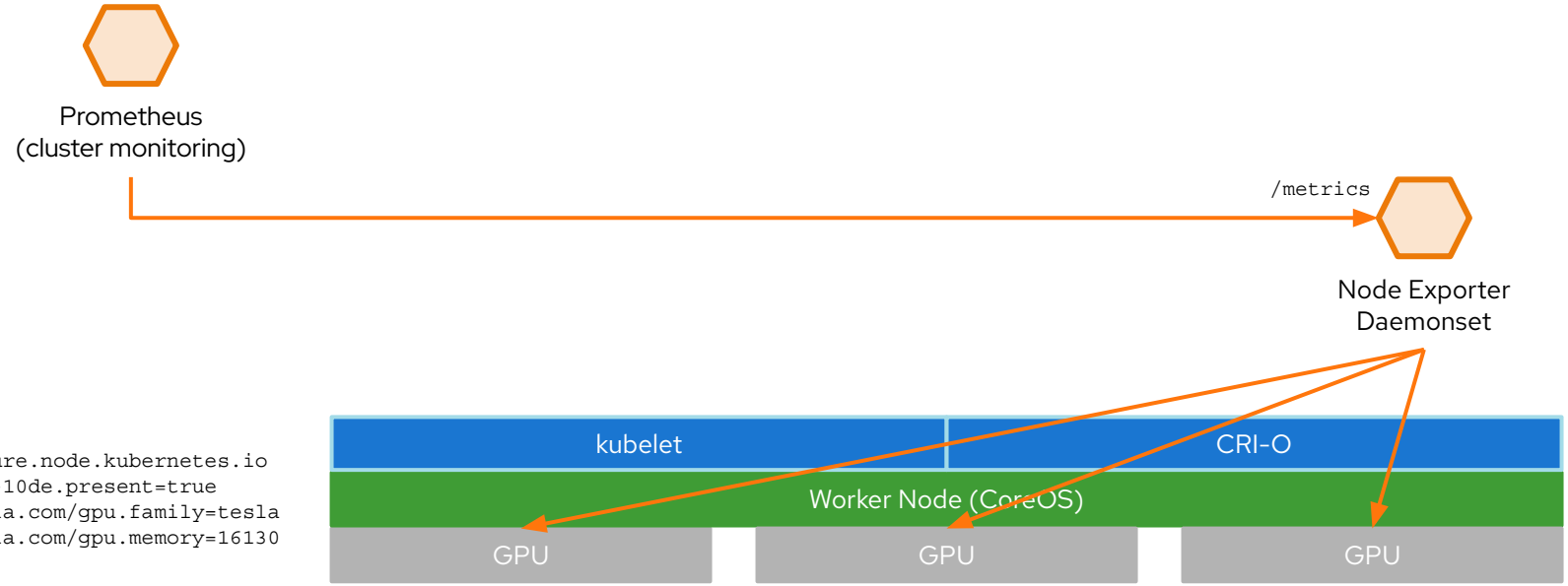

```
feature.node.kubernetes.io
/pci-10de.present=true
nvidia.com/gpu.family=tesla
nvidia.com/gpu.memory=16130
...
```

Device Plugin informs kubelet of resource details



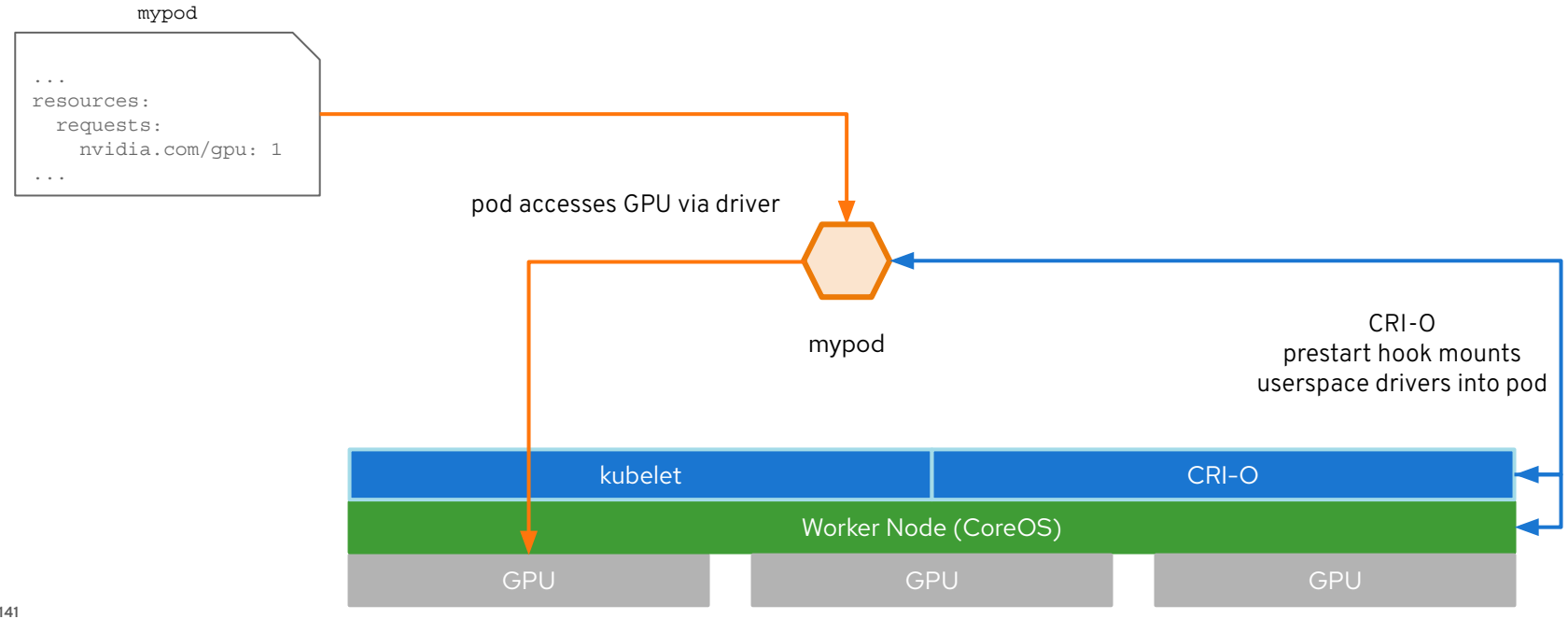

```
feature.node.kubernetes.io
/pci-10de.present=true
nvidia.com/gpu.family=tesla
nvidia.com/gpu.memory=16130
...
```

Node Exporter provides metrics on GPU




```
feature.node.kubernetes.io
/pci-10de.present=true
nvidia.com/gpu.family=tesla
nvidia.com/gpu.memory=16130
...
```

GPU workload deployment



Load Balancing and DNS with OpenShift IPI

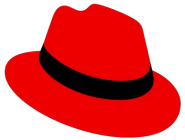
For physical, OSP, RHV, and vSphere IPI deployments

Red Hat

On-prem OpenShift IPI DNS and Load Balancer

- OpenShift 4.2, with OpenStack IPI, introduced a new way of doing DNS and load balancing for the `api`, `api-int`, DNS, and `*.apps` (Ingress) endpoints
  - OCP 4.4 added RHV IPI
  - OCP 4.5 added vSphere IPI
  - OCP 4.6 added physical IPI
- This method was originally used by the Kubernetes-native Infrastructure concept when creating bare metal clusters

mDNS

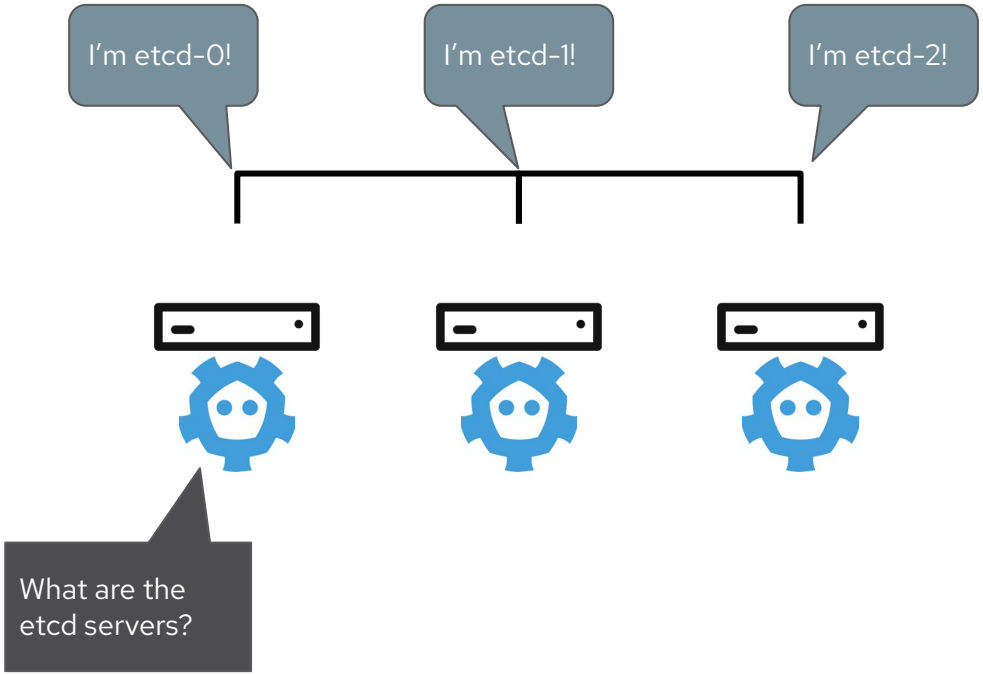


Red Hat OpenShift 4

Excruciating detail: https://github.com/openshift/installer/blob/master/docs/design/baremetal/networking-infrastructure.md

mDNS with CoreDNS

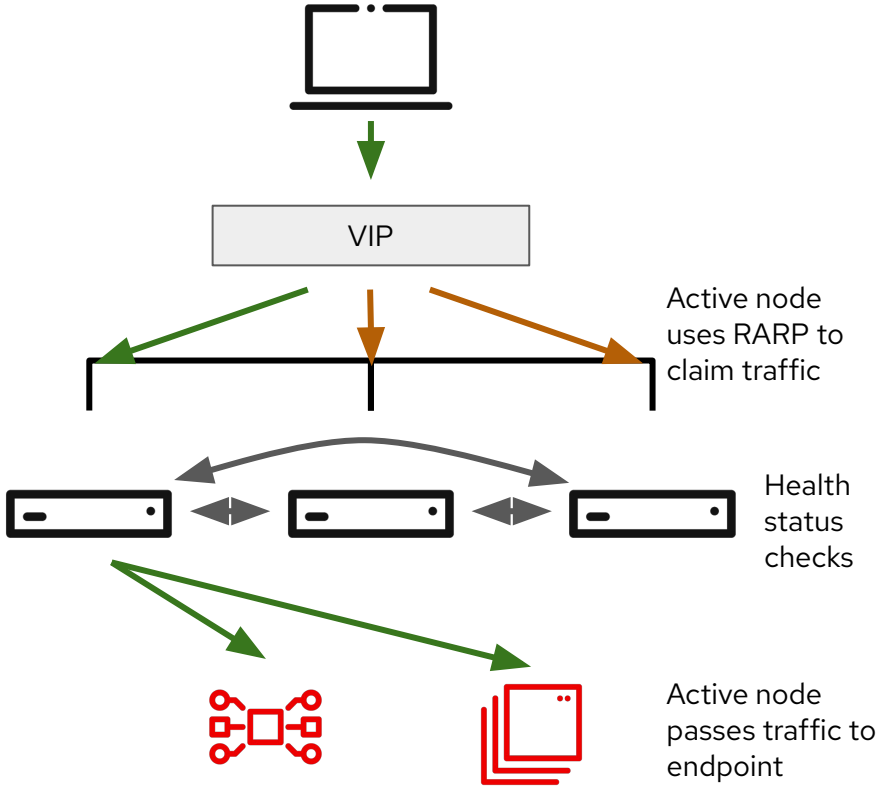
- CoreDNS is used by Kubernetes (and OpenShift) for **internal service discovery**
  - Not used for node discovery
- Multicast DNS (mDNS) works by sending DNS packets, using UDP, to a specific multicast address
  - mDNS hosts listen on this address and respond to queries
- mDNS in OpenShift
  - Nodes publish IP address/hostname for themselves to local mDNS responder
  - mDNS responder on each node replies with local value
- DNS SRV records are not used for etcd in OCP 4.4 and later



Red Hat

keepalived

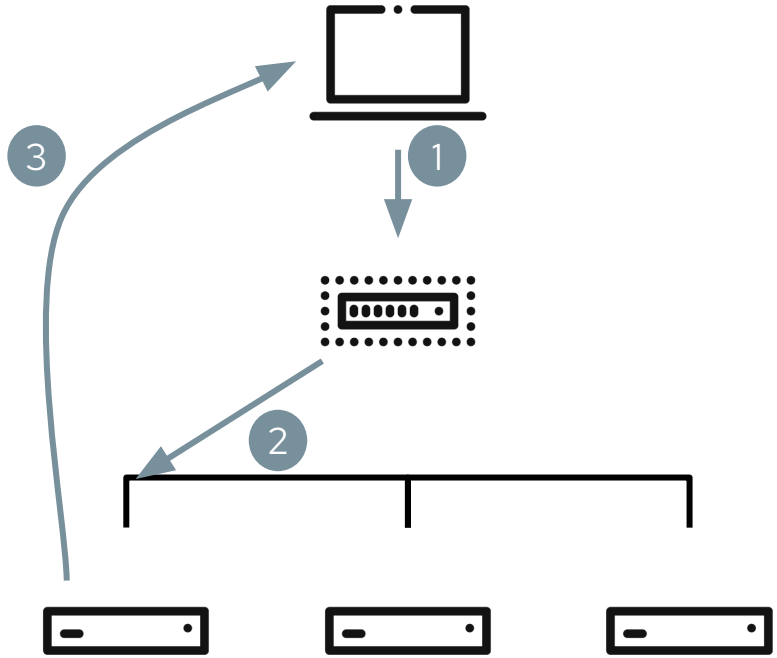
- Used to ensure that the API and Ingress (*.apps) Virtual IPs (VIP) are always available
- Utilizes Virtual Router Redundancy Protocol (VRRP) to determine node health and elect an IP owner
  - Only one host owns the IP at any time
  - All nodes have equal priority
  - Failover can take several seconds
- Node health is checked every one second
  - Separate checks for each service (API, Ingress/*.apps)
- ARP is used to associate the VIP with the owner node's interface



Red Hat

API load balancer

1) Client creates a new request to `api.cluster-name.domain.name`
2) HAProxy on the node actively hosting the API IP address (as determined by `keepalived`) load balances across control plane nodes using round robin
3) The connection is forwarded to the chosen control plane node, which responds directly to the client, a.k.a. "direct return"

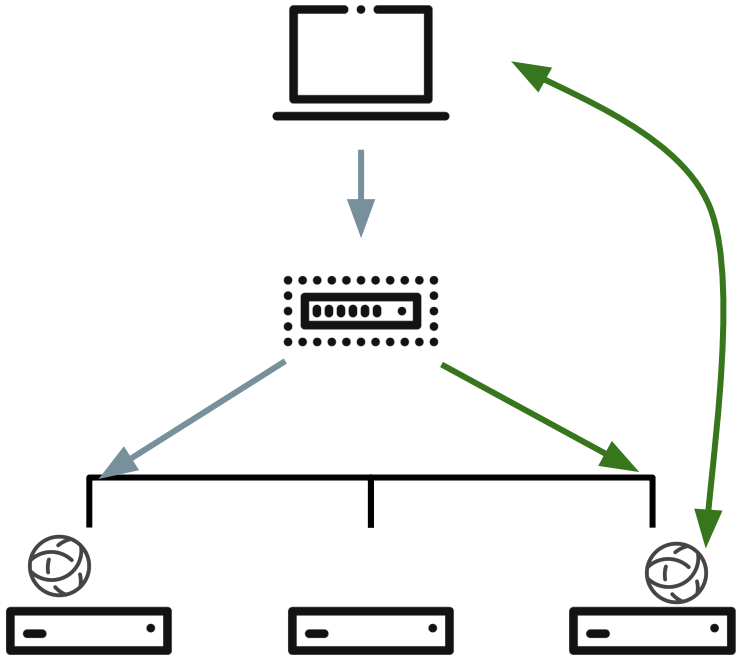


Control plane nodes

Red Hat

Ingress load balancer

- The VIP, managed by Keepalived, will only be hosted on nodes which have a Router instance
  - Nodes without a Router continue to participate in the VRRP domain, but fail the check script, so are ineligible for hosting the VIP
- Traffic destined for the `*.apps` Ingress VIP will be passed directly to the Router instance



Worker / Infra nodes

Requirements and limitations

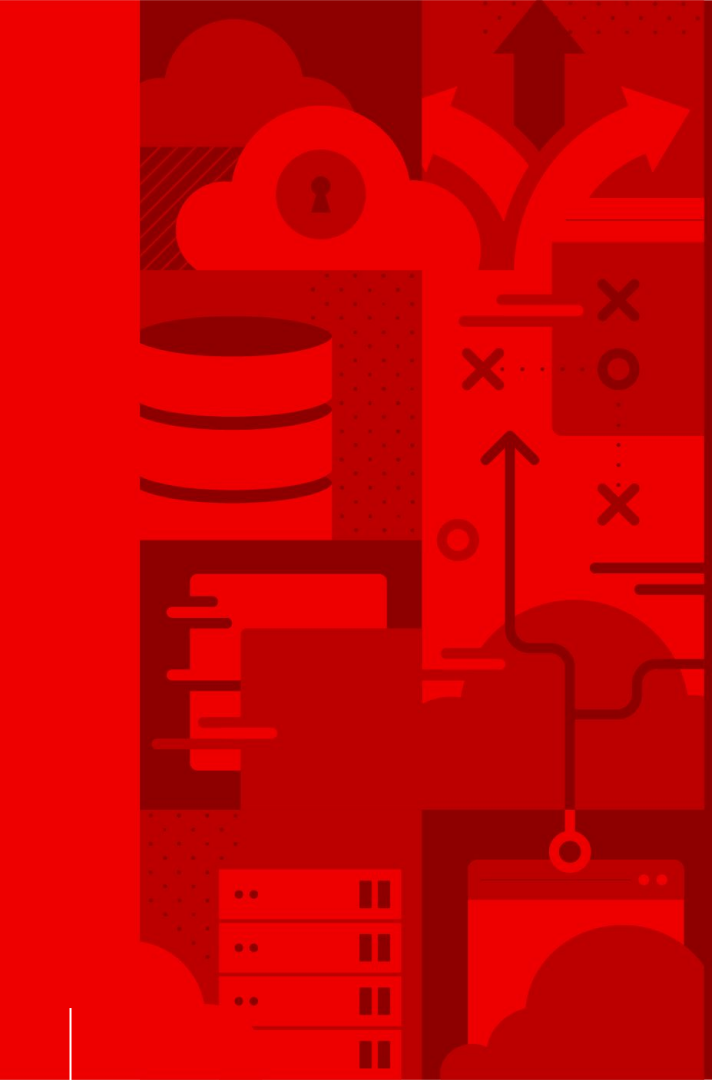
1) Multicast is required for the Keepalived (VRRP) and mDNS configuration used
2) VRRP needs layer 2 adjacency to function
   a) All control plane nodes must be on the same subnet
   b) All worker nodes capable of hosting a router instance must be on the same subnet
   c) The VIPs must be on the same subnet as the hosts
3) Ingress (*.apps) throughput is limited to a single node
4) Keepalived failover will result in disconnected sessions, e.g. `oc logs -f <pod>` will terminate
   a) Failover may take several seconds
5) There cannot be more than 119 on-prem IPI cluster instances on the same L2 domain
   a) Each cluster uses two VRRP IDs (API, ingress)
   b) The function used to generate router IDs returns values of 1-239
   c) There is no collision detection between clusters for the VRRP ID
   d) The chance of collision goes up as additional clusters are deployed

Alternatives

"I don't like this," "I can't use this," and/or "this does not meet my needs". What other options are there?

- Ingress
  - 3rd party partners, such as F5 and Citrix, have certified Operators that are capable of replacing the Ingress solution as a day 2 operation
- API
  - There is no supported way of replacing the API Keepalived + HAProxy configuration
- DNS
  - There is no supported way of replacing mDNS in this configuration
- DHCP
  - DHCP is required for all IPI deployments, there is no supported way of using static IPs with IPI

Remember that IPI is opinionated. If the customer's needs cannot be met by the IPI config, and it's not an option to reconfigure within the scope of supported options, then UPI is the solution. Machine API integration can be deployed as a day 2 operation for node scaling.



Ευχαριστώ

Red Hat