OpenShift

Container Platform

Distance Learning

ARCHITECTURAL OVERVIEW

Alfred Bach
Principal Solution Architect


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos

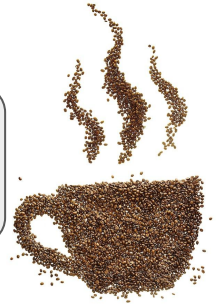

facebook.com/redhatinc


twitter.com/RedHat

Red Hat

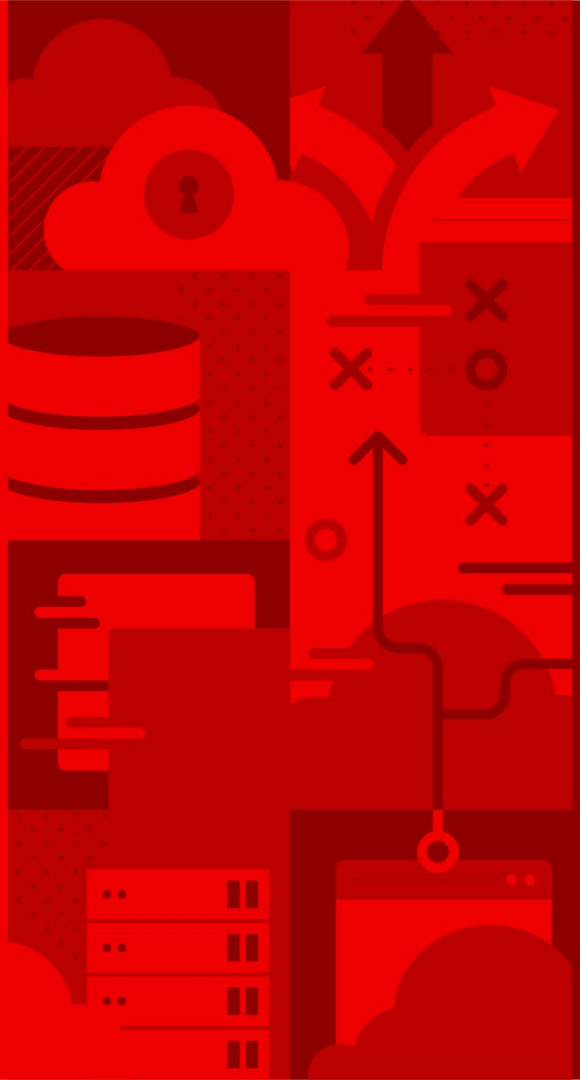
OpenShift Architecture

Part 1
(16:15 - 17:00)



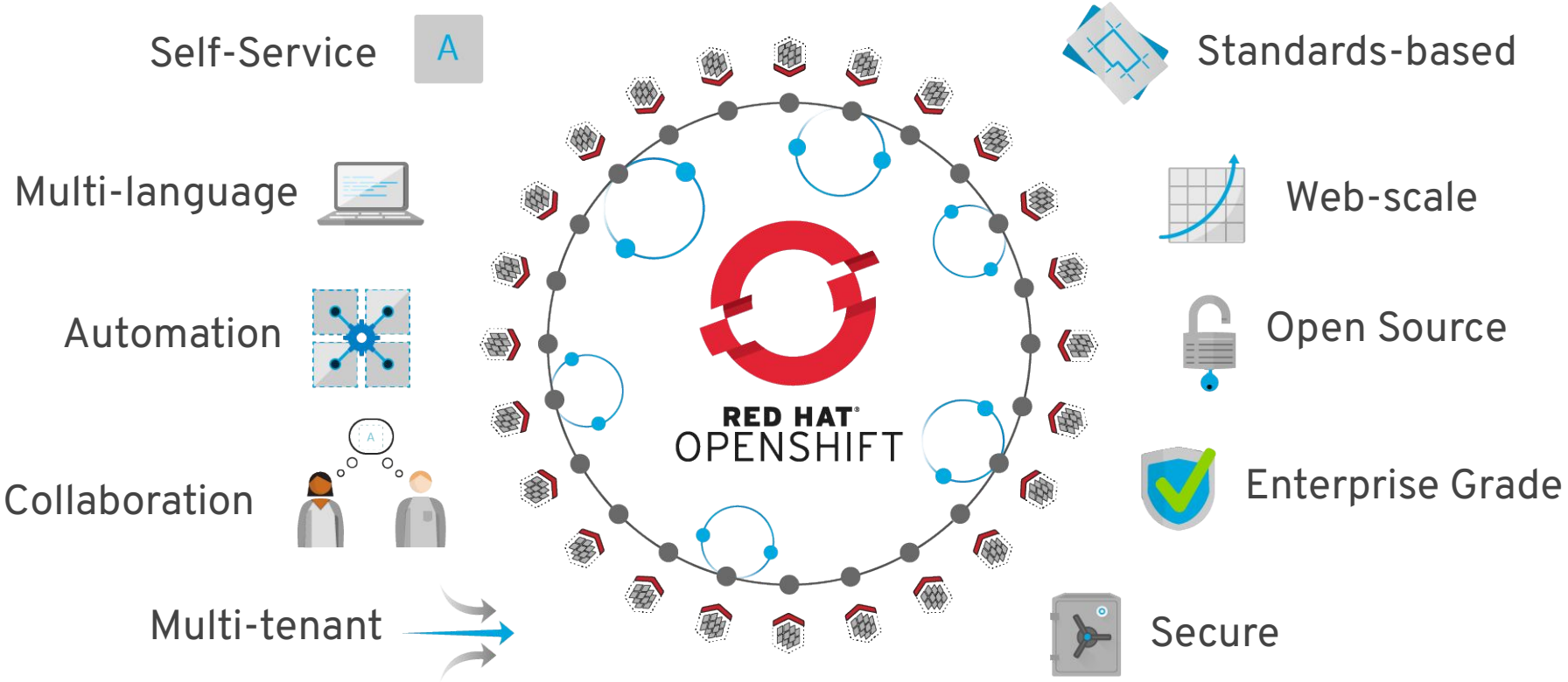
Part 2
(17:05 - 18:00)

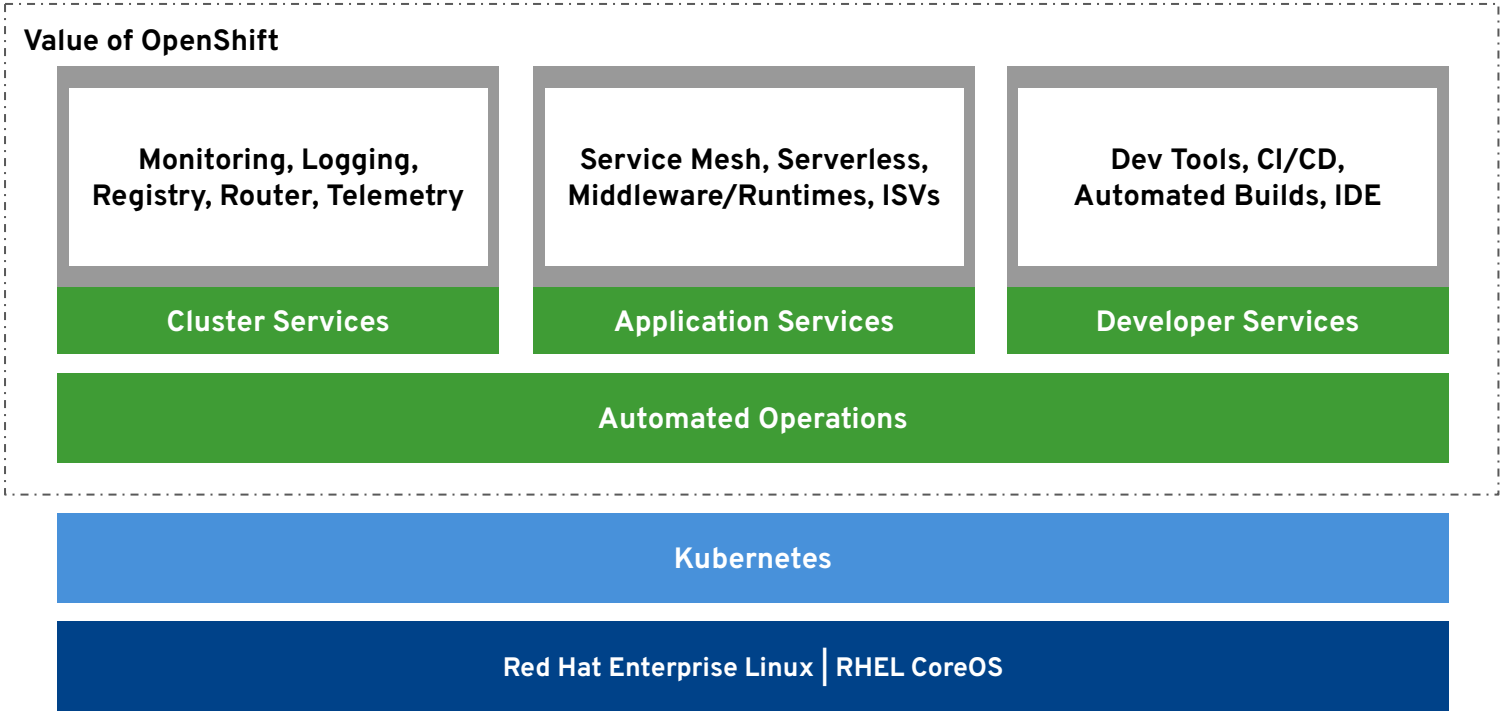
Part 3
Day 2 (16:05 - 17:00)



Functional overview

Red Hat

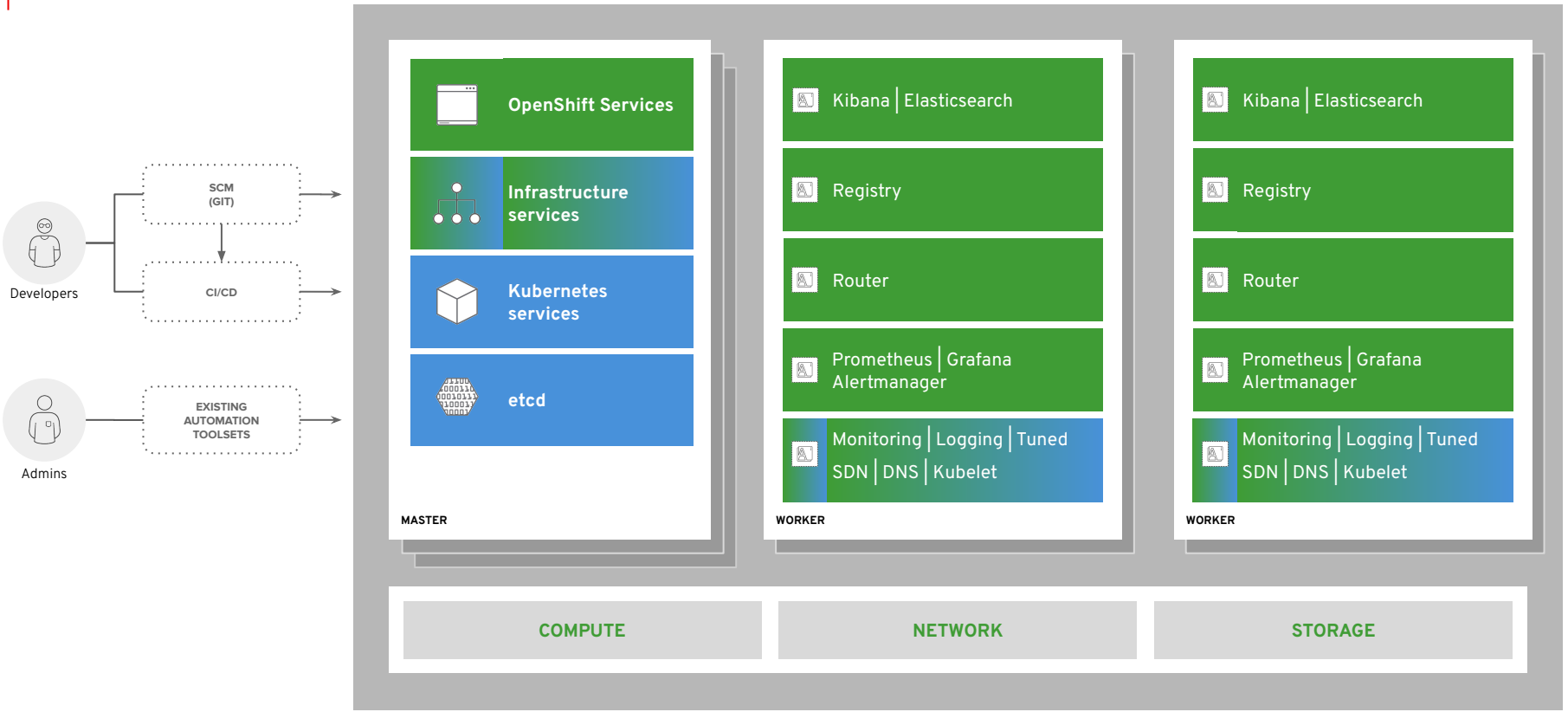

Self-Service
Multi-language
Automation
Collaboration
Multi-tenant
RED HAT
OPENSHIFT
Standards-based
Web-scale
Open Source
Enterprise Grade
Secure



Best IT Ops Experience

CaaS ⟷ PaaS ⟷ FaaS

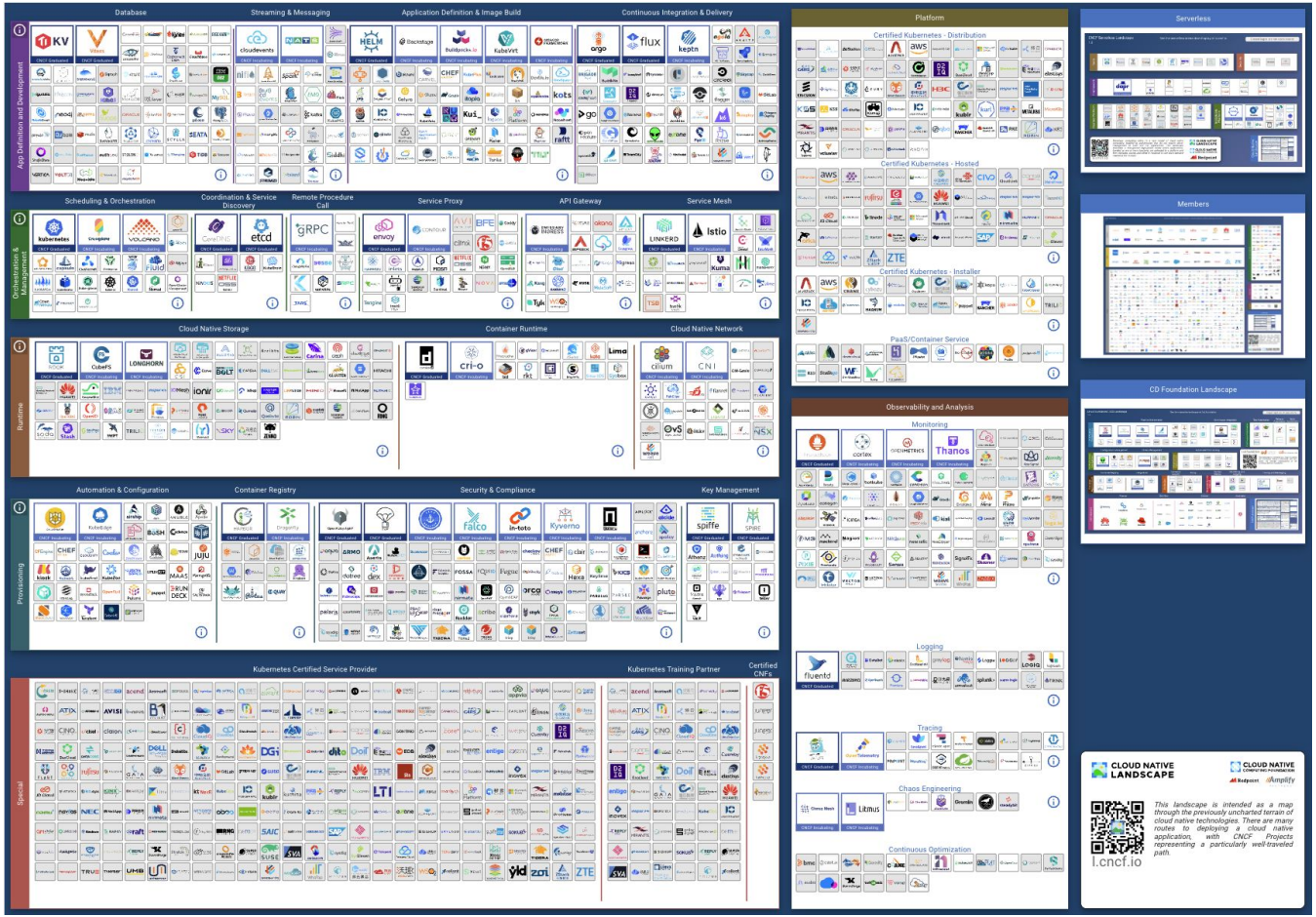
Best Developer Experience


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

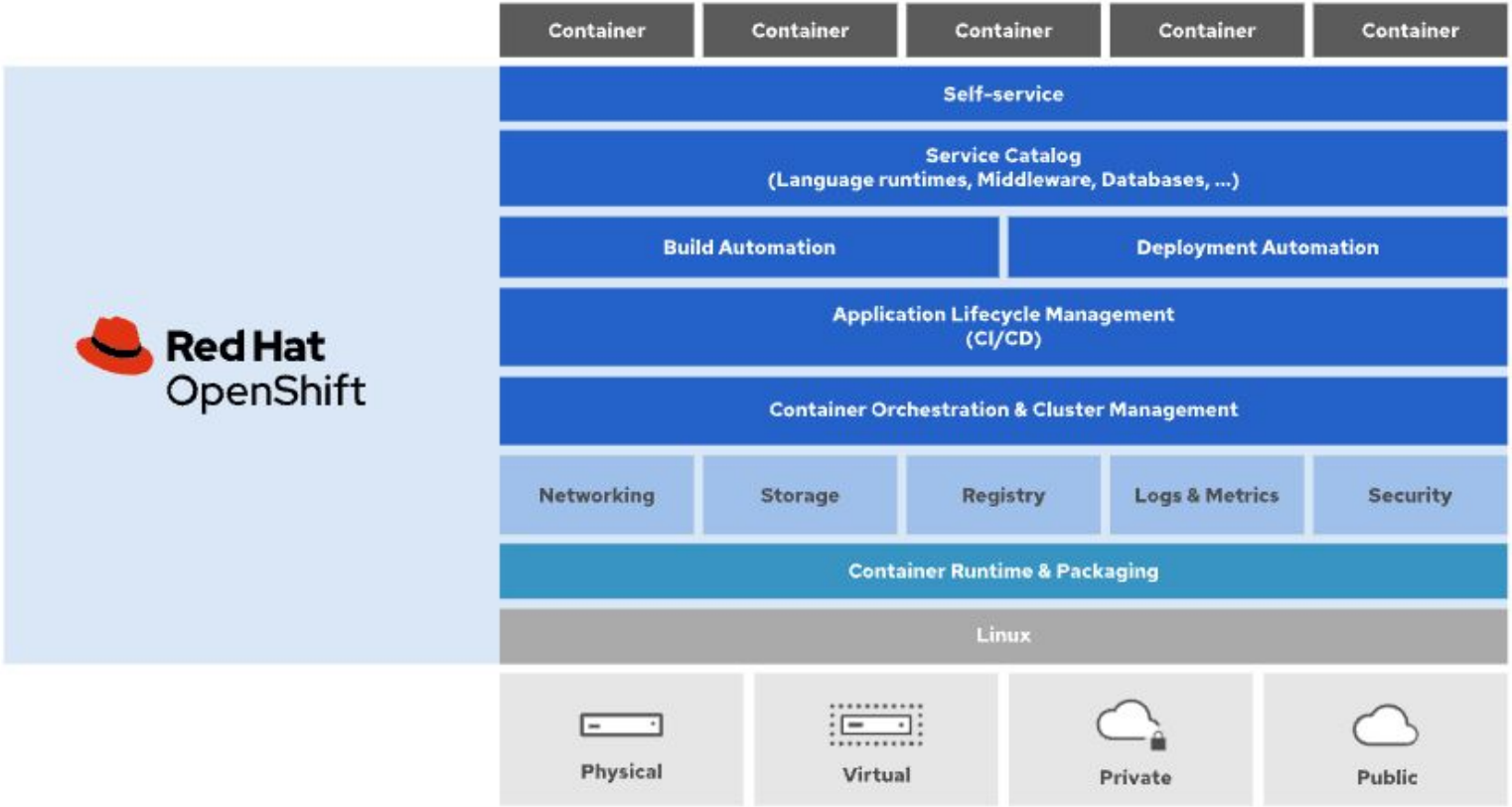
Red Hat

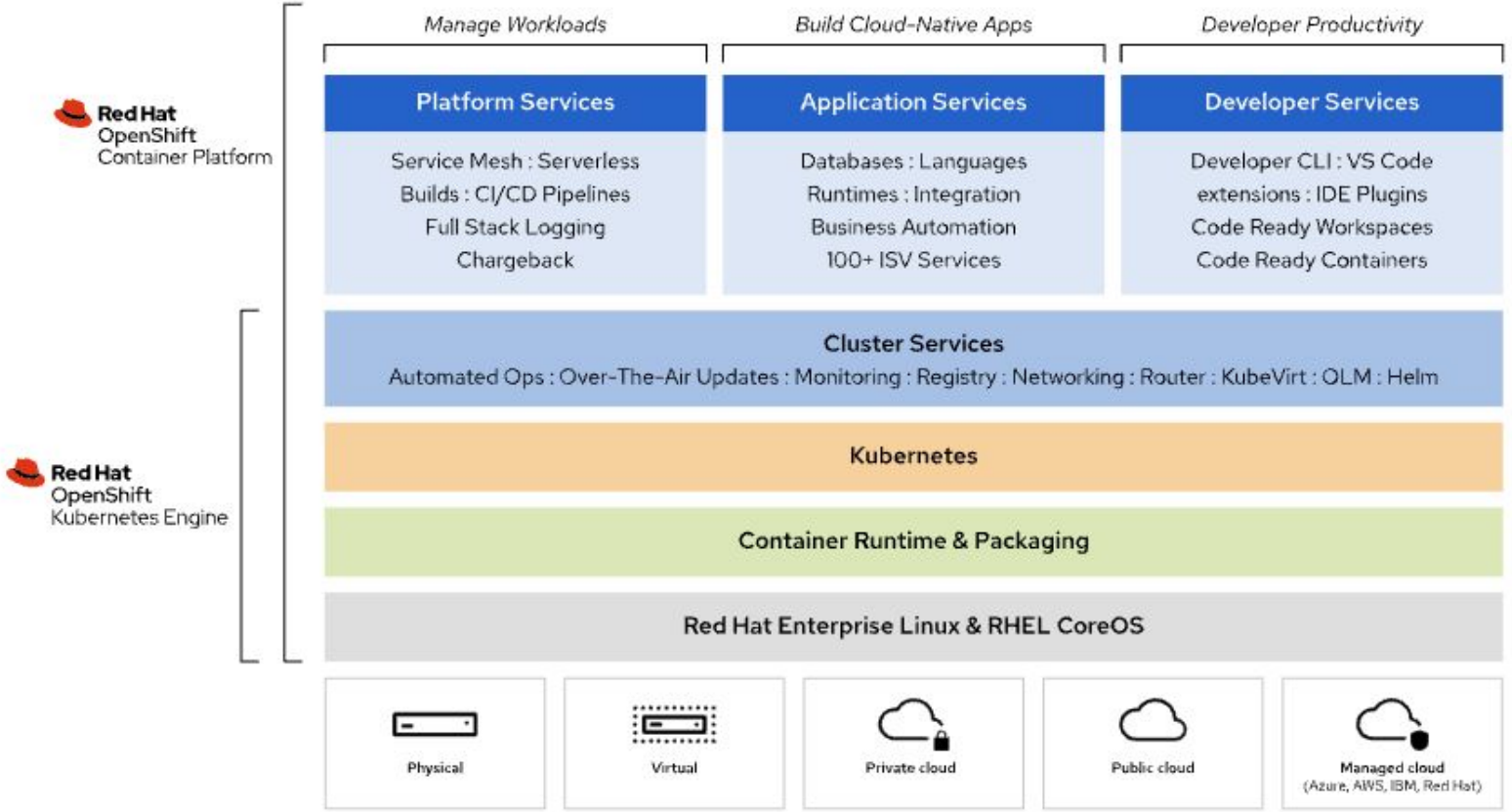
CLOUD NATIVE LANDSCAPE

CNCF Ecosystem Slide

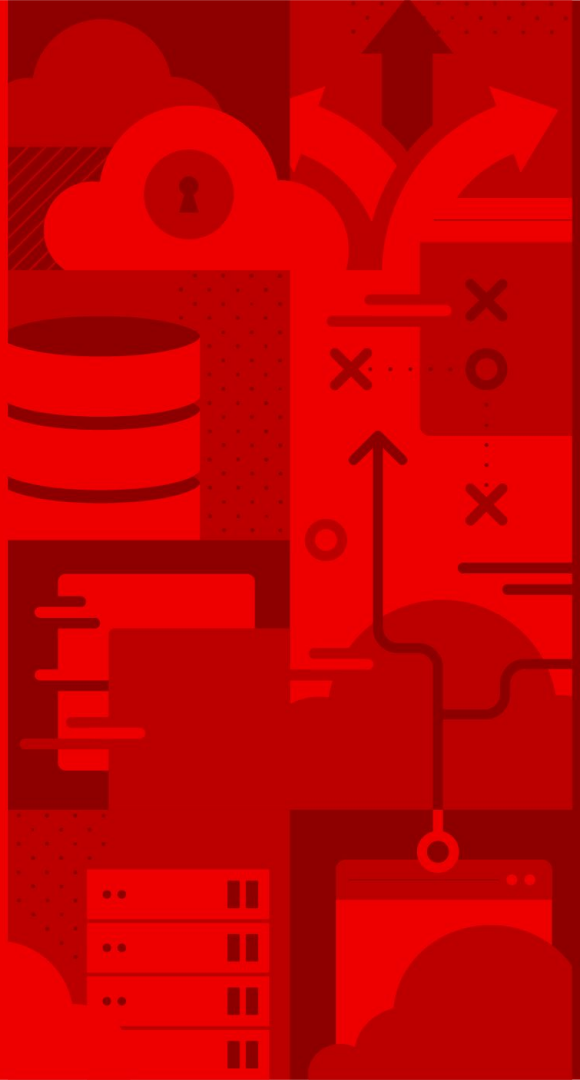


Red Hat


Container
Container
Container
Container
Container
Self-service
Service Catalog
(Language runtimes, Middleware, Databases, ...)
Build Automation
Deployment Automation
Application Lifecycle Management
(CI/CD)
Container Orchestration & Cluster Management
Networking
Storage
Registry
Logs & Metrics
Security
Container Runtime & Packaging
Linux
Red Hat
OpenShift
Physical
Virtual
Private
Public


Red Hat OpenShift Container Platform
Red Hat OpenShift Kubernetes Engine
Manage Workloads
Build Cloud-Native Apps
Developer Productivity
Platform Services
Service Mesh : Serverless
Builds : CI/CD Pipelines
Full Stack Logging
Chargeback
Application Services
Databases : Languages
Runtimes : Integration
Business Automation
100+ ISV Services
Developer Services
Developer CLI : VS Code
extensions : IDE Plugins
Code Ready Workspaces
Code Ready Containers
Cluster Services
Automated Ops : Over-The-Air Updates : Monitoring : Registry : Networking : Router : KubeVirt : OLM : Helm
Kubernetes
Container Runtime & Packaging
Red Hat Enterprise Linux & RHEL CoreOS
Physical
Virtual
Private cloud
Public cloud
Managed cloud
(Azure, AWS, IBM, Red Hat)

Red Hat

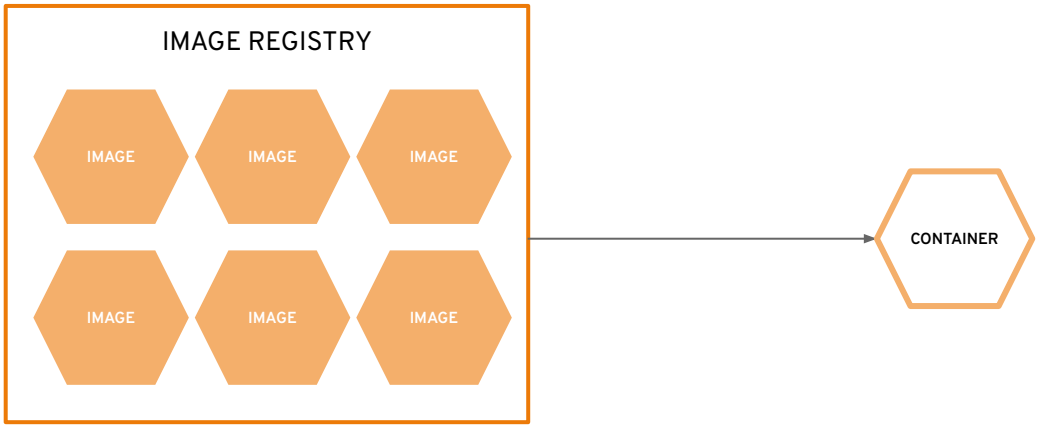


OpenShift and Kubernetes core concepts

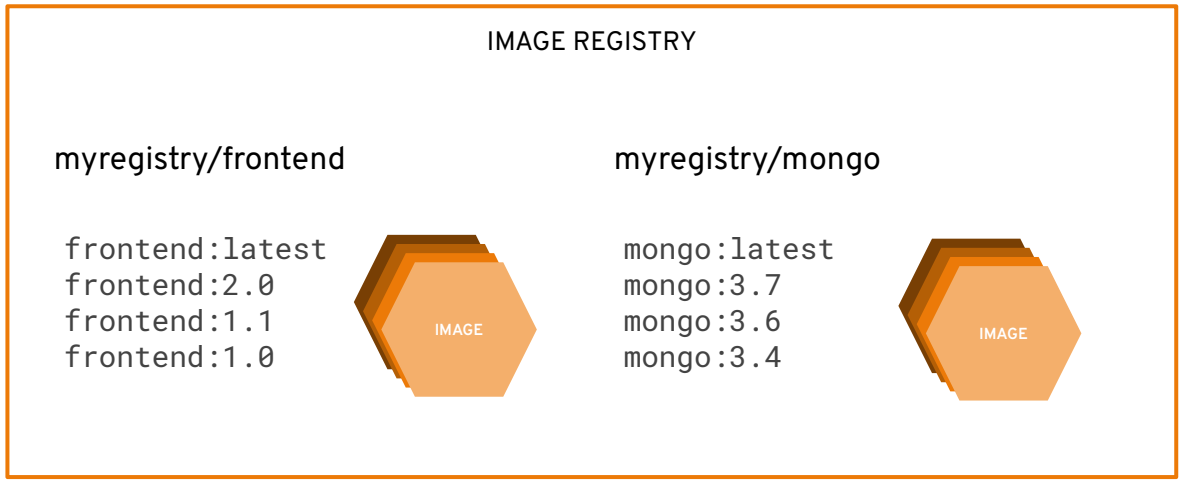
a container is the smallest compute unit

containers are created from container images

container images are stored in an image registry

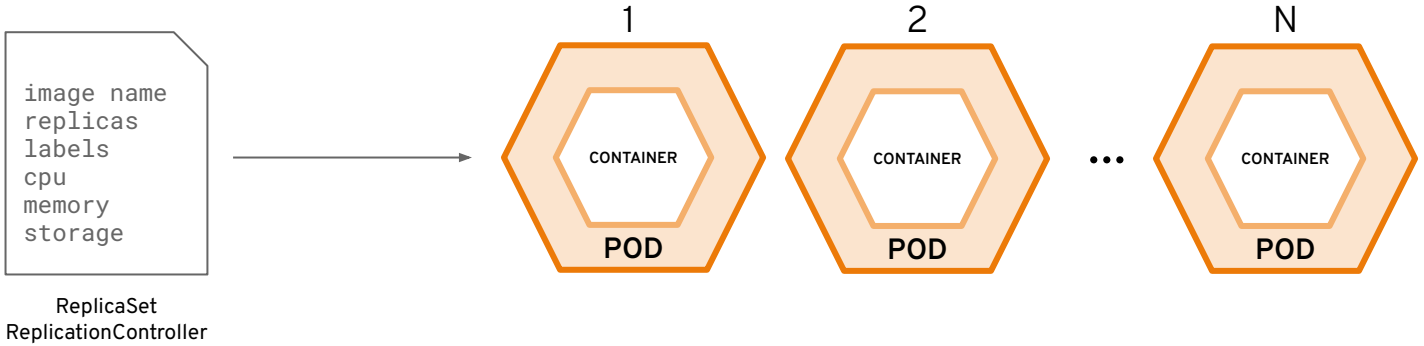


an image repository contains all versions of an image in the image registry



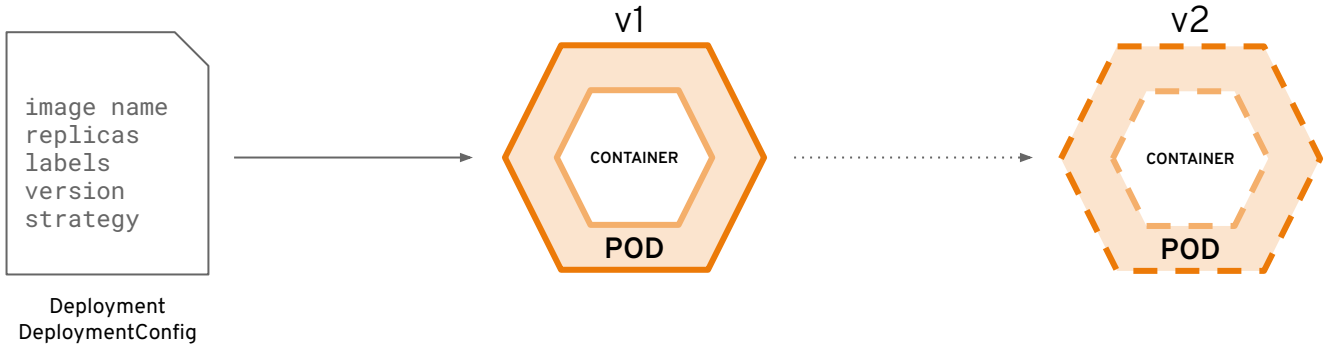
containers are wrapped in pods which are units of deployment and management

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

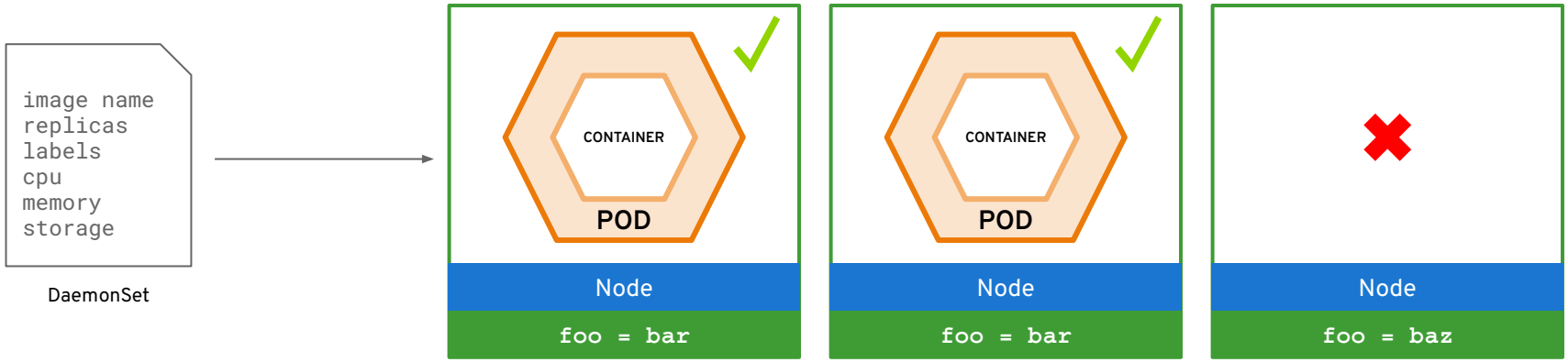


Red Hat

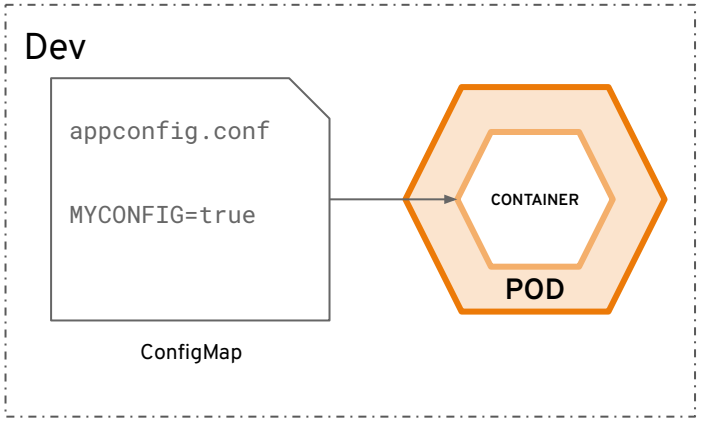
`Deployments` and `DeploymentConfigurations` define how to roll out new versions of Pods

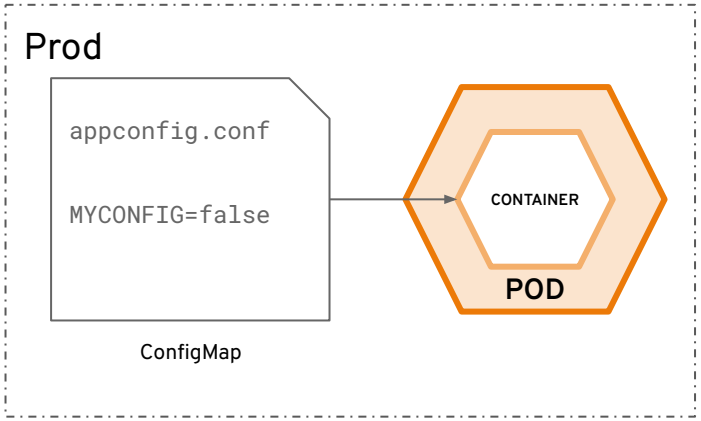


a daemonset ensures that all (or some) nodes run a copy of a pod

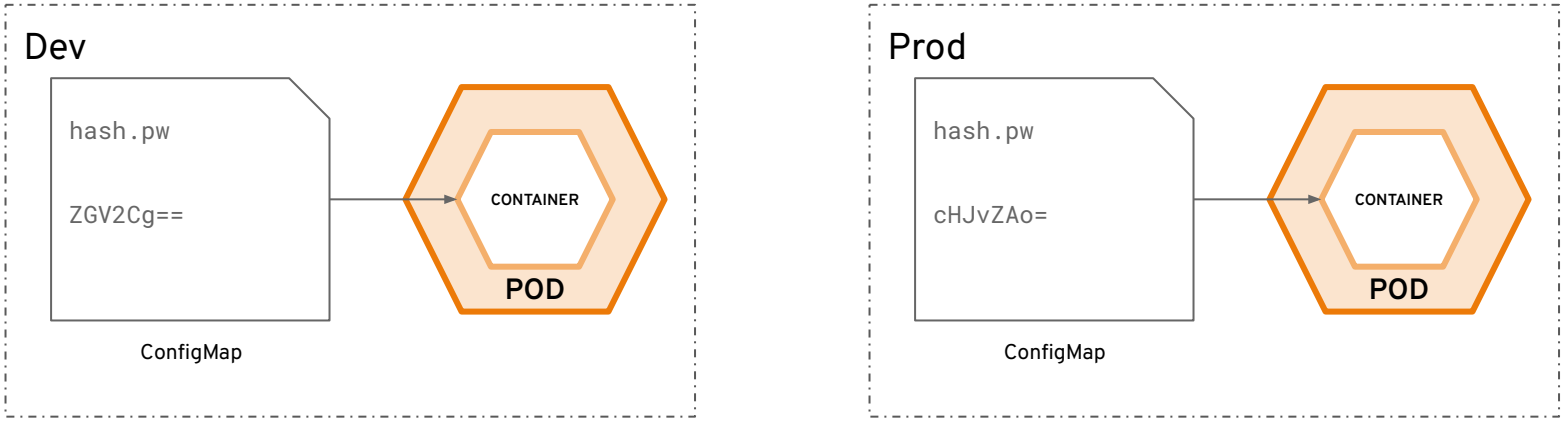


`configmaps` allow you to decouple configuration artifacts from image content



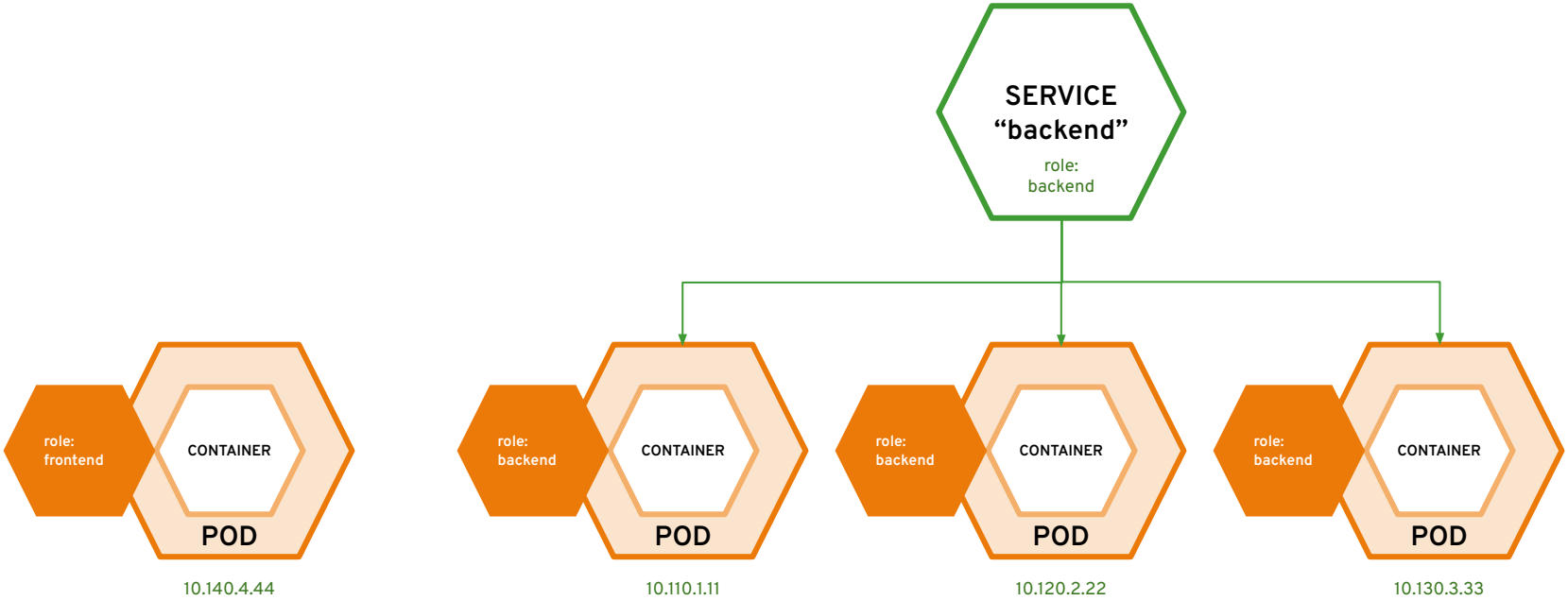


`secrets` provide a mechanism to hold sensitive information such as passwords

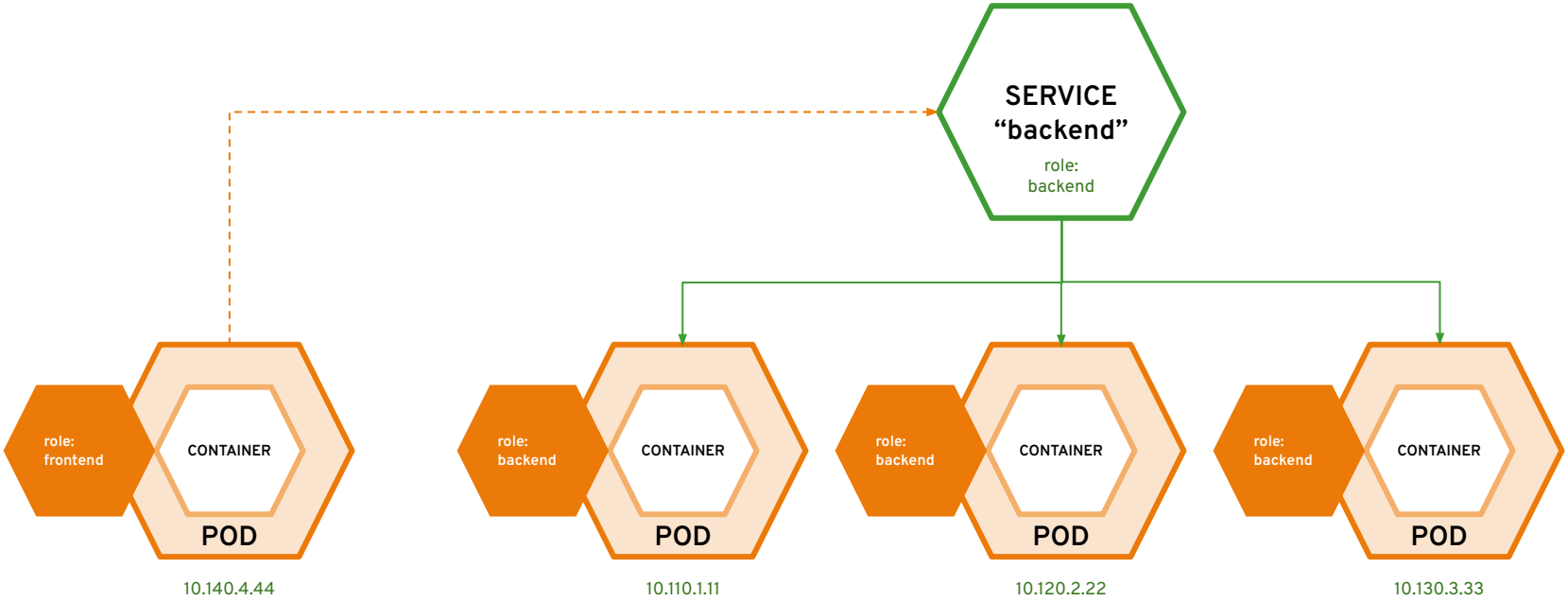


The etcd datastore can be encrypted for additional security
https://docs.openshift.com/container-platform/4.6/security/encrypting-etcd.html

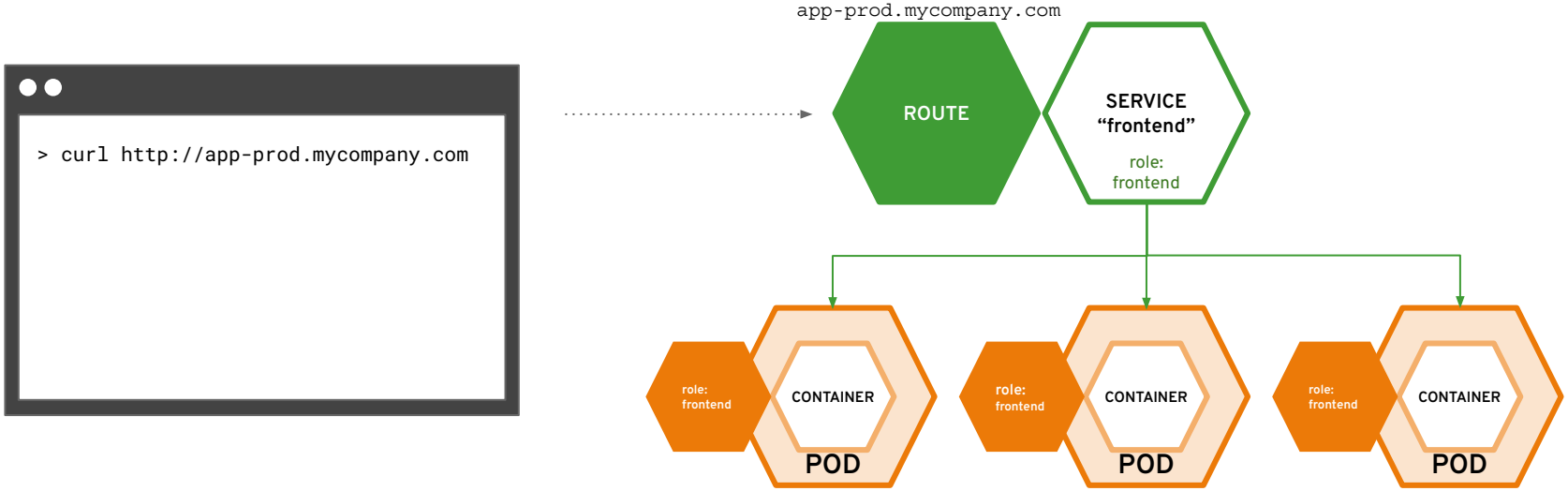
services provide internal load-balancing and service discovery across pods



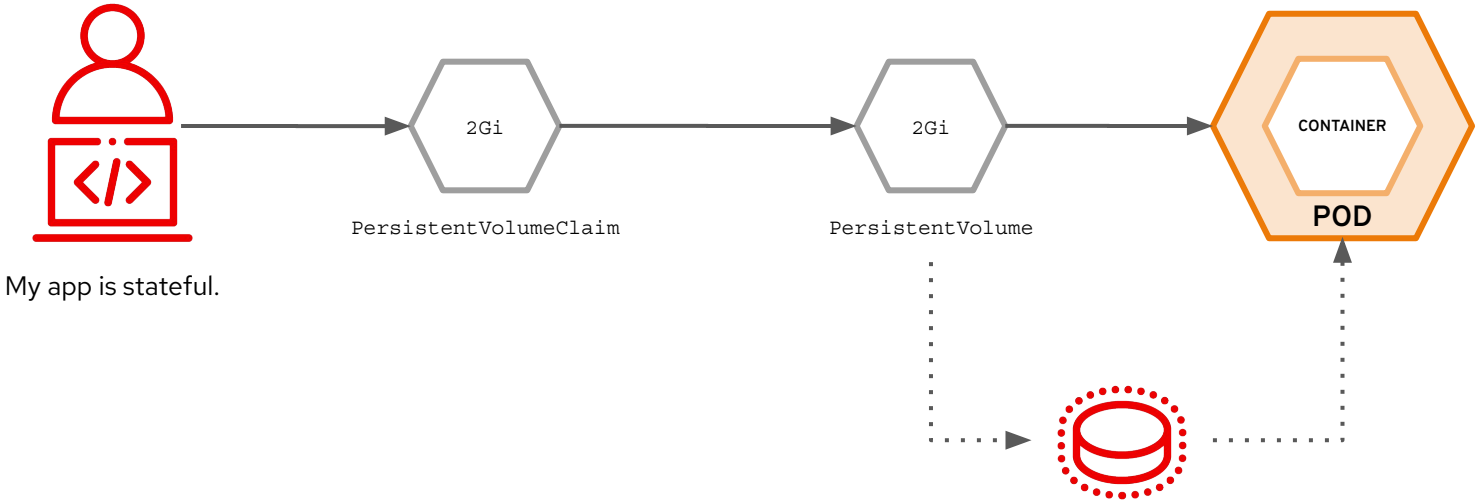
apps can talk to each other via services



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

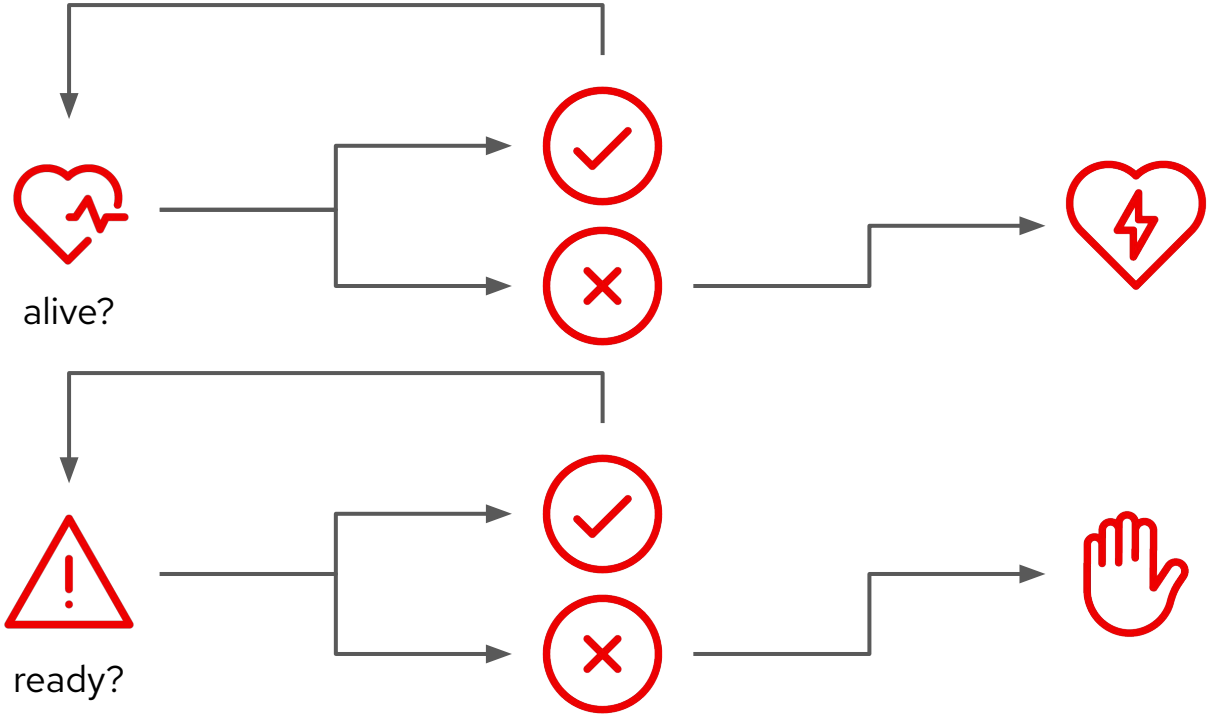


Persistent Volume and Claims

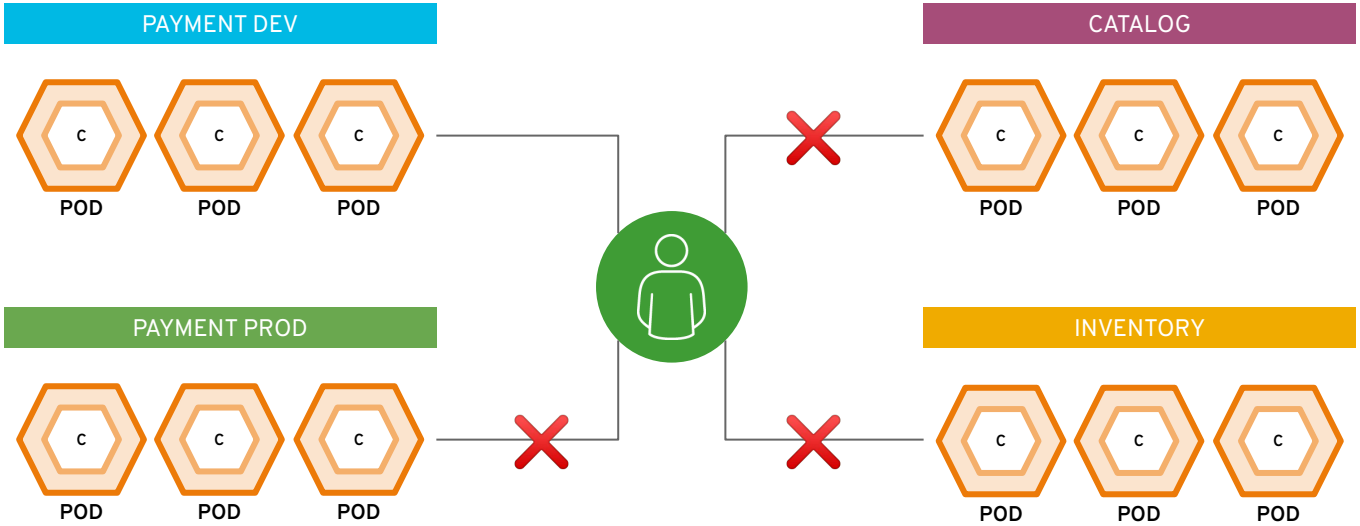


My app is stateful.

Liveness and Readiness



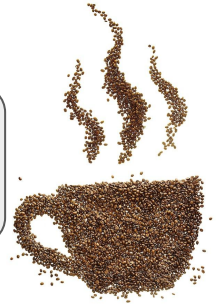
projects isolate apps across environments, teams, groups and departments



Red Hat

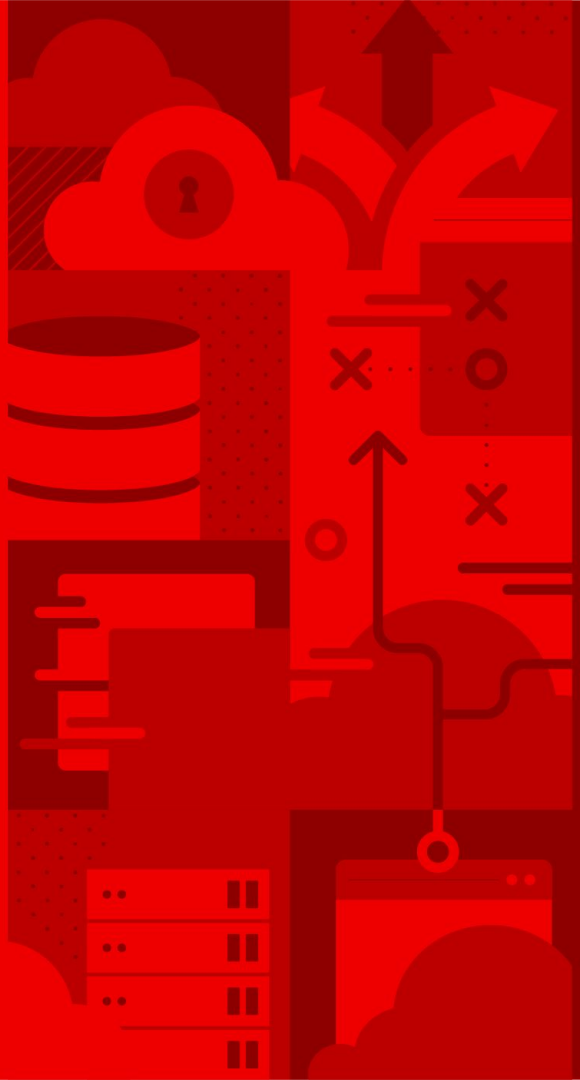
OpenShift Architecture

Part 1
(16:15 - 17:00)



Part 2
(17:05 - 18:00)

Part 3
Day 2 (16:05 - 17:00)



OpenShift 4 Architecture

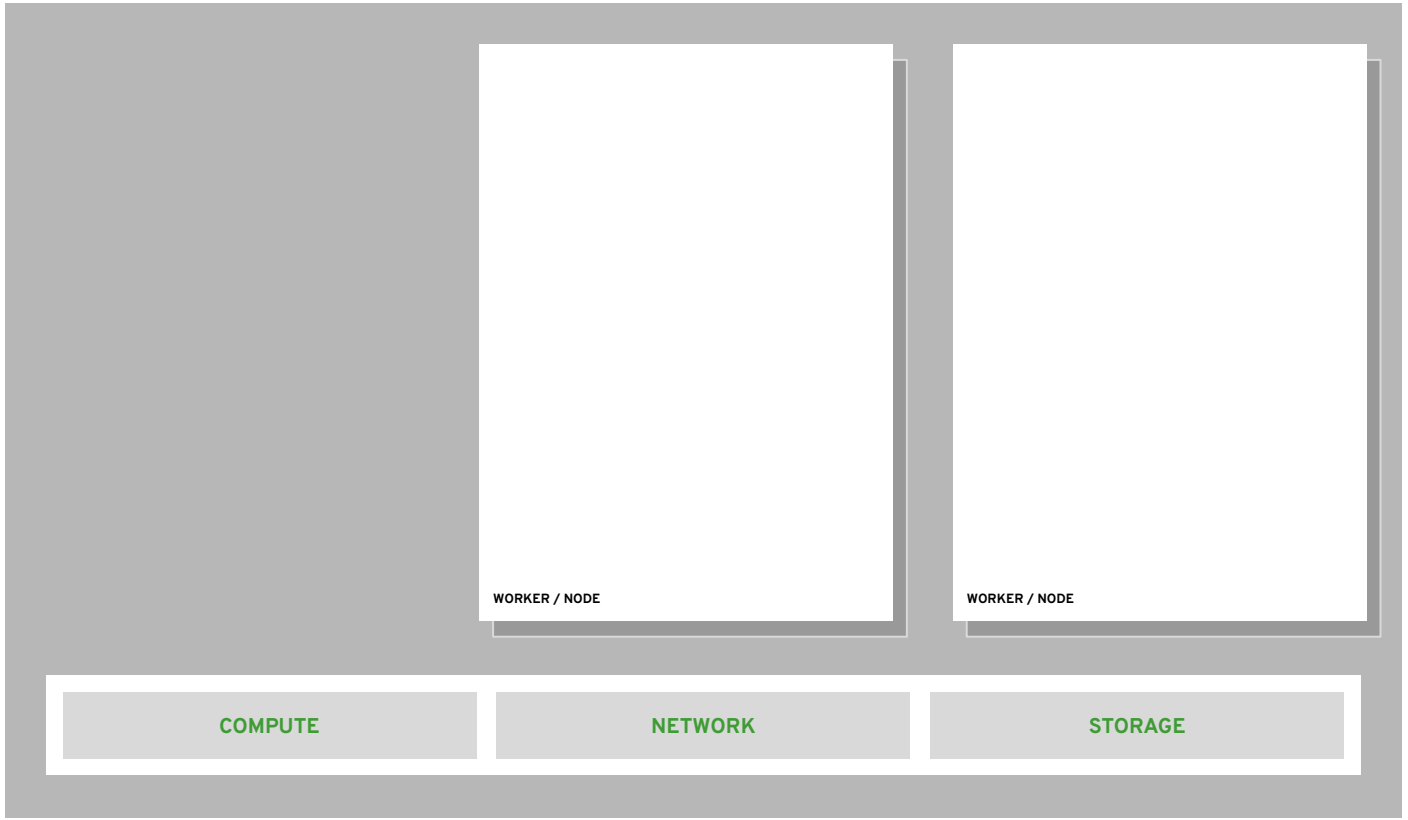
your choice of infrastructure

COMPUTE

NETWORK

STORAGE

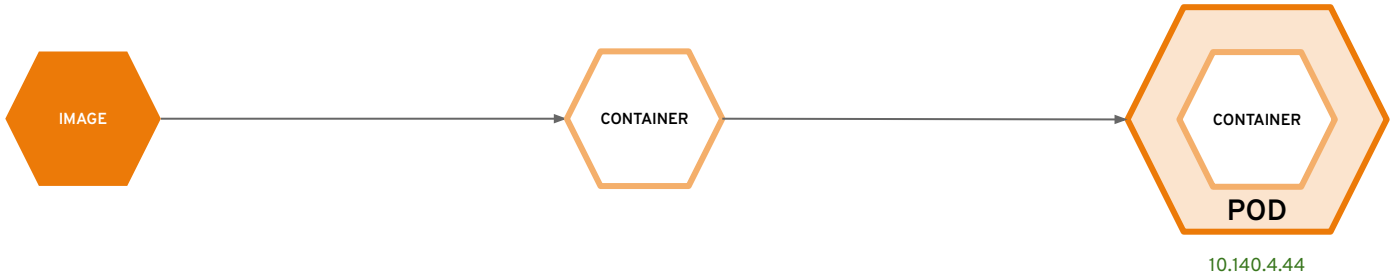
workers run workloads



masters are the control plane

Red Hat

everything runs in pods



state of everything

core kubernetes components

Introducing the Declarative Architecture of Kubernetes

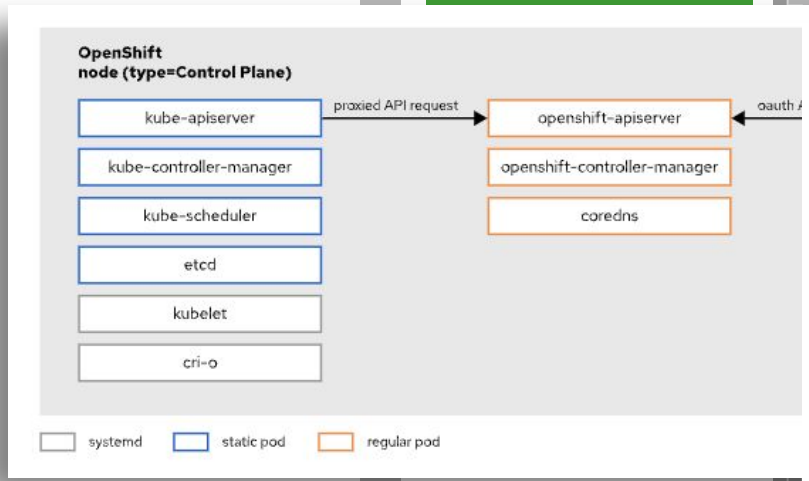
The architecture of OpenShift is based on the declarative nature of Kubernetes. Most system administrators are used to imperative architectures, where you perform actions that indirectly change the state of the system, such as starting and stopping containers on a given server. In a declarative architecture, you change the state of the system and the system updates itself to comply with the new state. For example, with Kubernetes, you define a pod resource that specifies that a certain container should run under specific conditions. Then Kubernetes finds a server (a node) that can run that container under these specific conditions.

Declarative architectures allow for self-optimizing and self-healing systems that are easier to manage than imperative architectures.

Kubernetes defines the state of its cluster, including the set of deployed applications, as a set of resources stored in the etcd database. Kubernetes also runs controllers that monitor these resources and compares them to the current state of the cluster. These controllers take any action necessary to reconcile the state of the cluster with the state of the resources, for example by finding a node with sufficient CPU capacity to start a new container from a new pod resource.

Kubernetes provides a REST API to manage these resources. All actions that an OpenShift user takes, either using the command-line interface or the web console, are performed by invoking this REST API.

core OpenShift components



Describing OpenShift Extensions

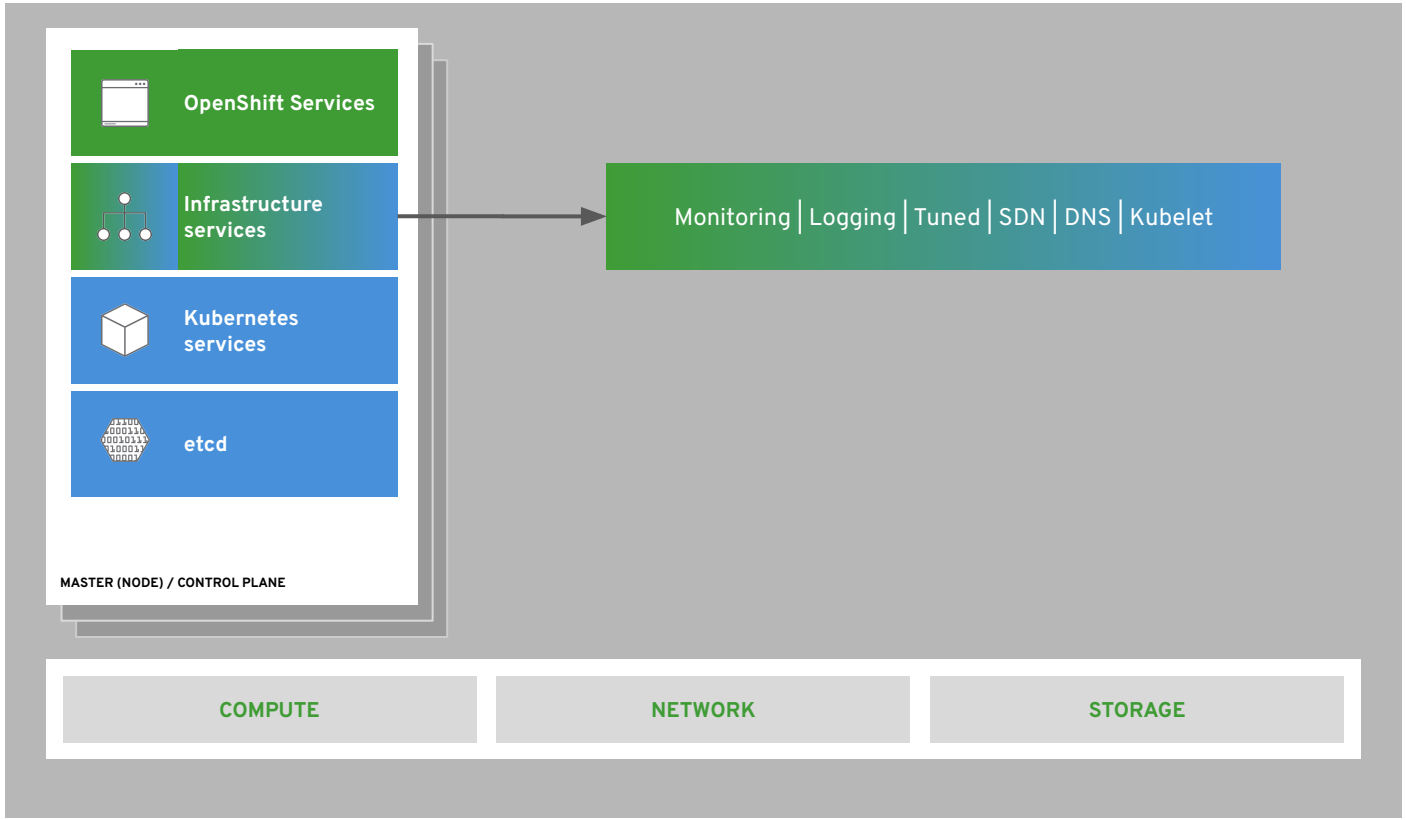
A lot of functionality from Kubernetes depends on external components, such as ingress controllers, storage plug-ins, network plug-ins, and authentication plug-ins. Similar to Linux distributions, there are many ways to build a Kubernetes distribution by picking and choosing different components.

A lot of functionality from Kubernetes also depends on extension APIs, such as access control and network isolation.

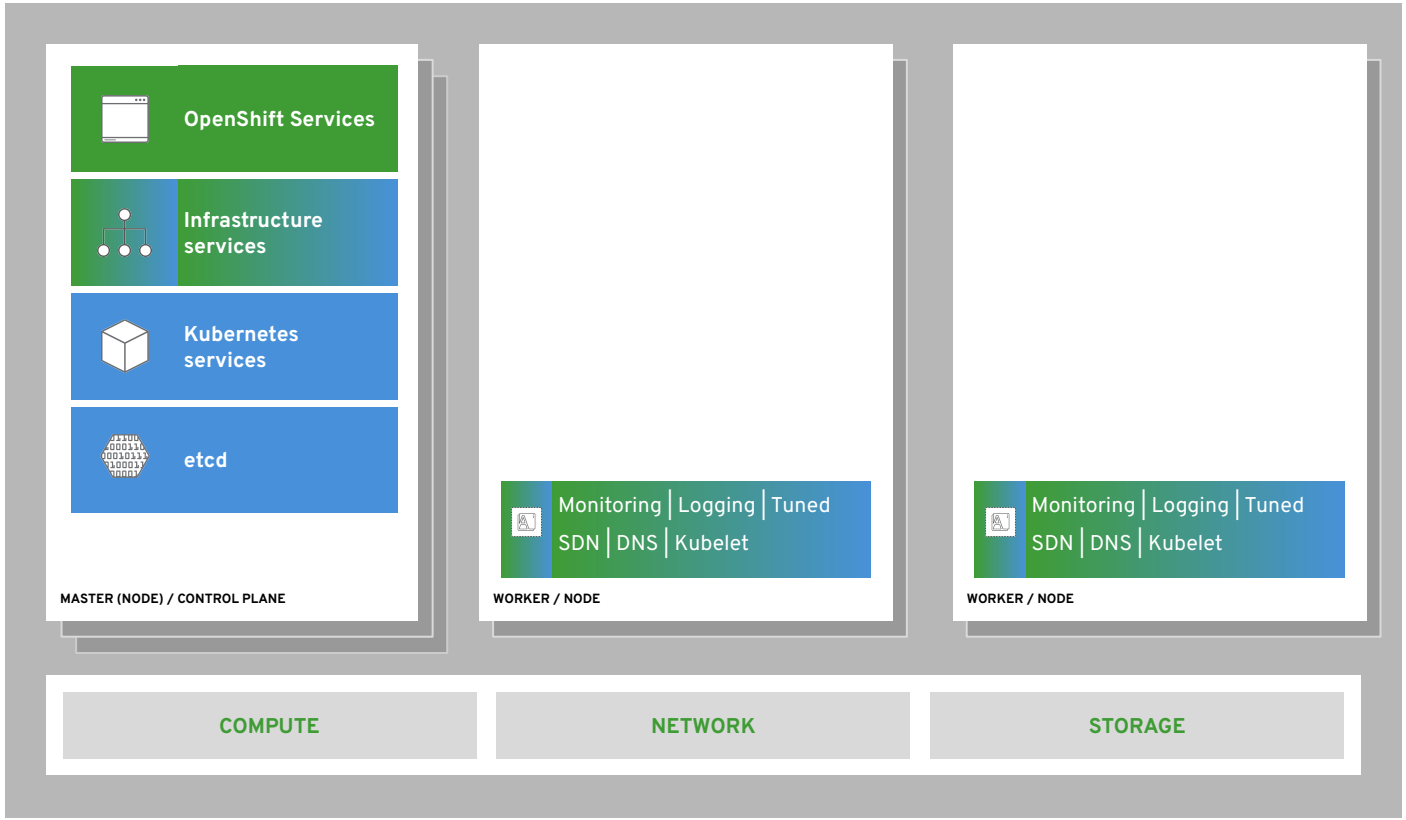
OpenShift is a Kubernetes distribution that provides many of these components already integrated and configured, and managed by operators. OpenShift also provides preinstalled applications, such as a container image registry and a web console, managed by operators.

at

internal and support infrastructure services

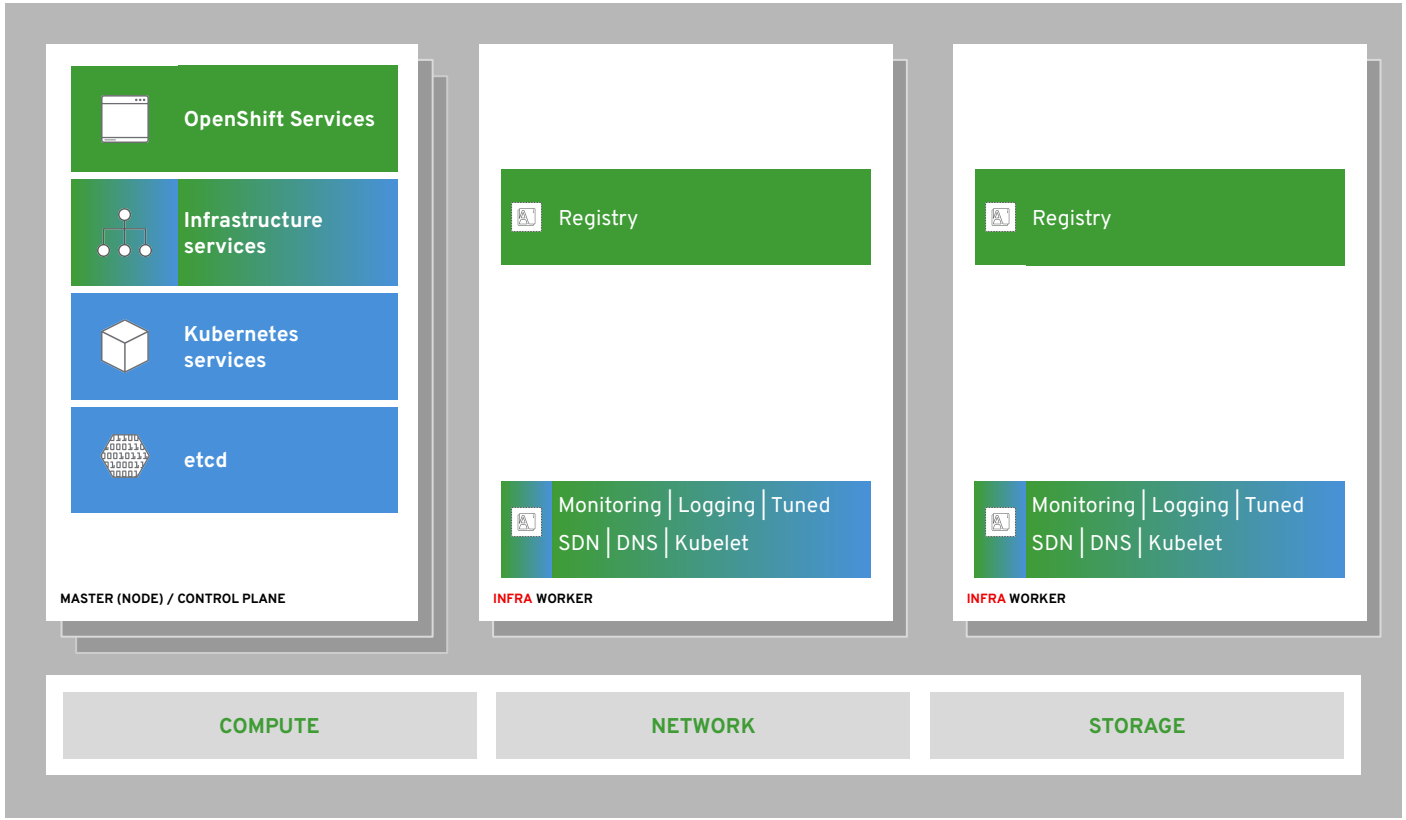


run on all hosts



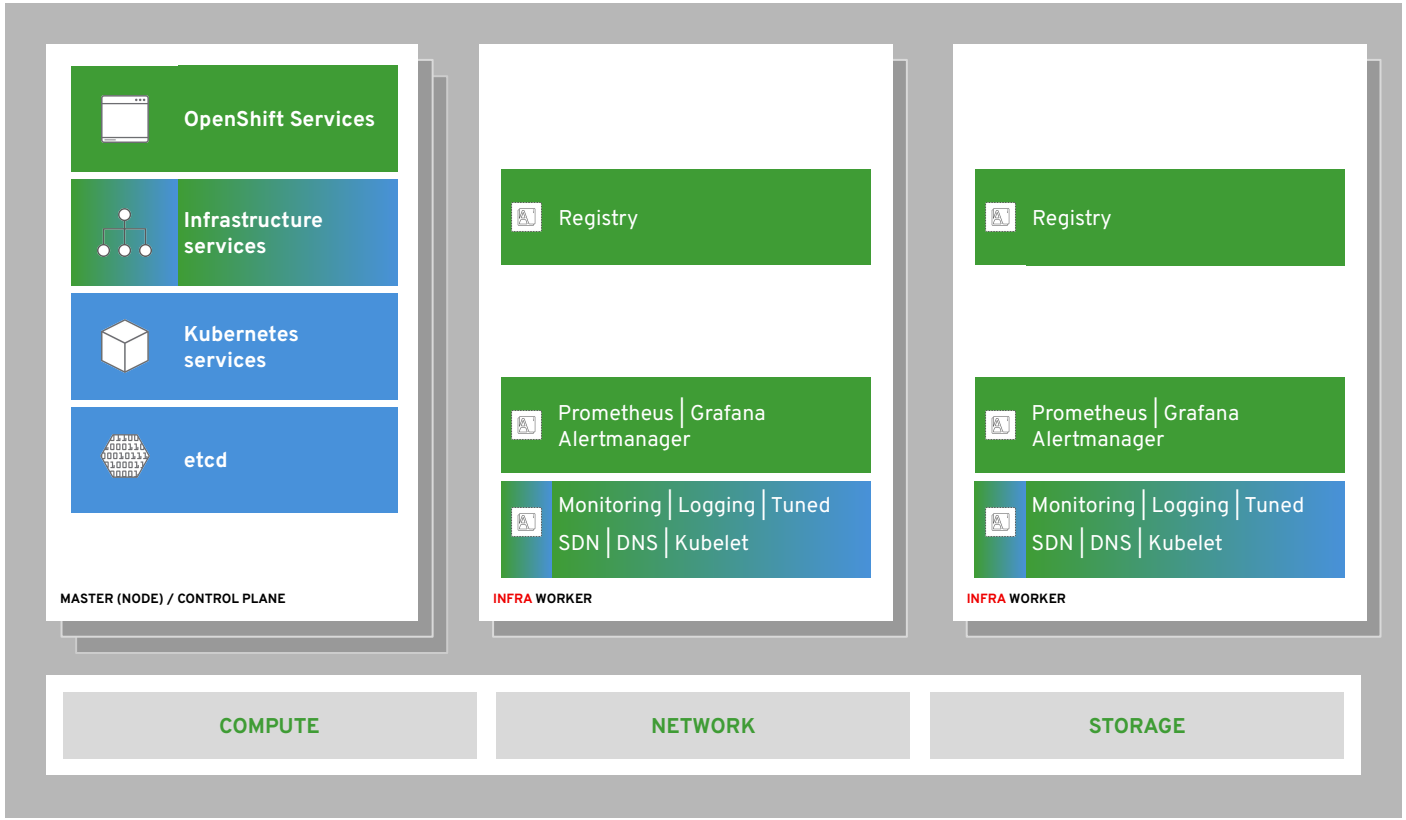
Red Hat

integrated image registry

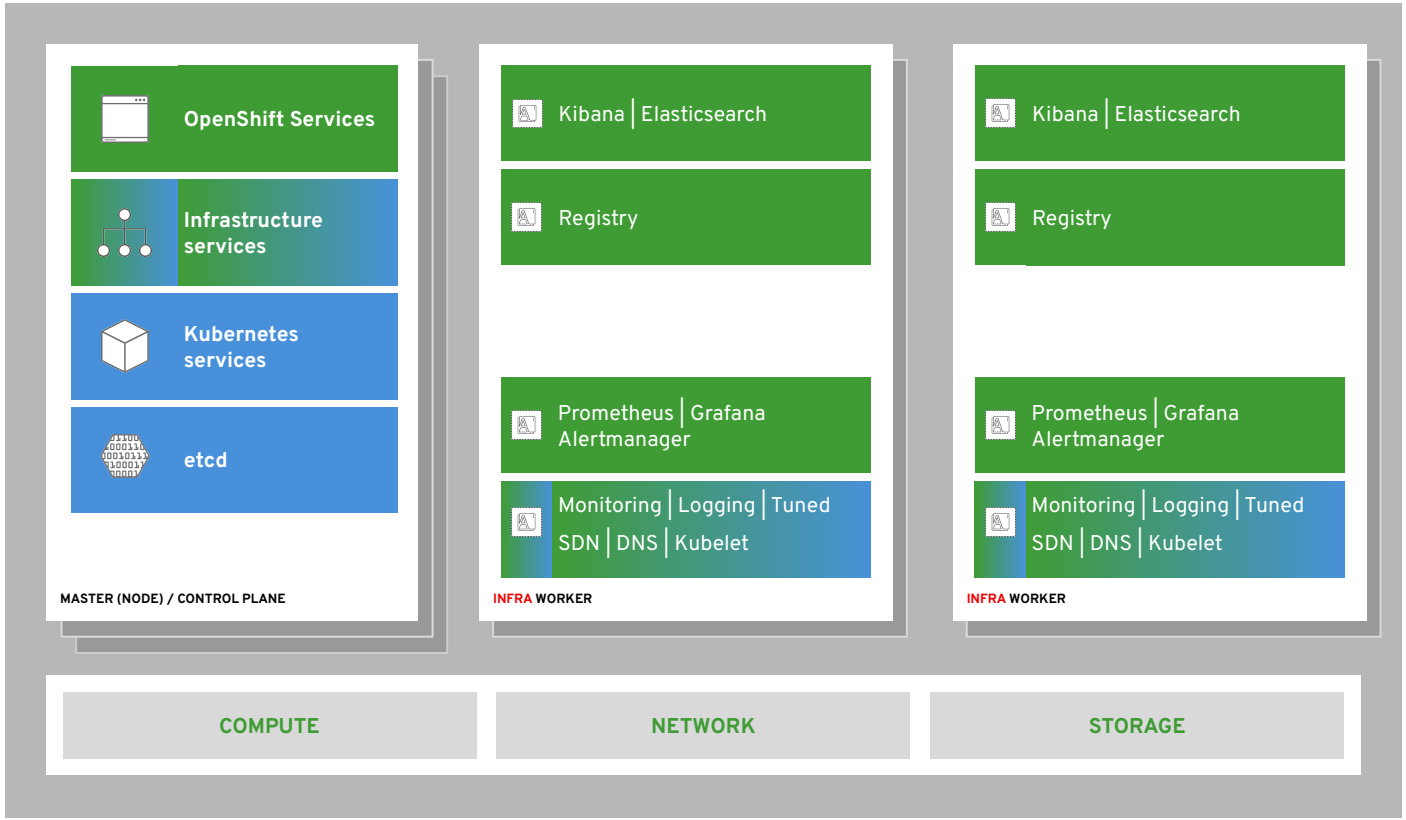


Red Hat

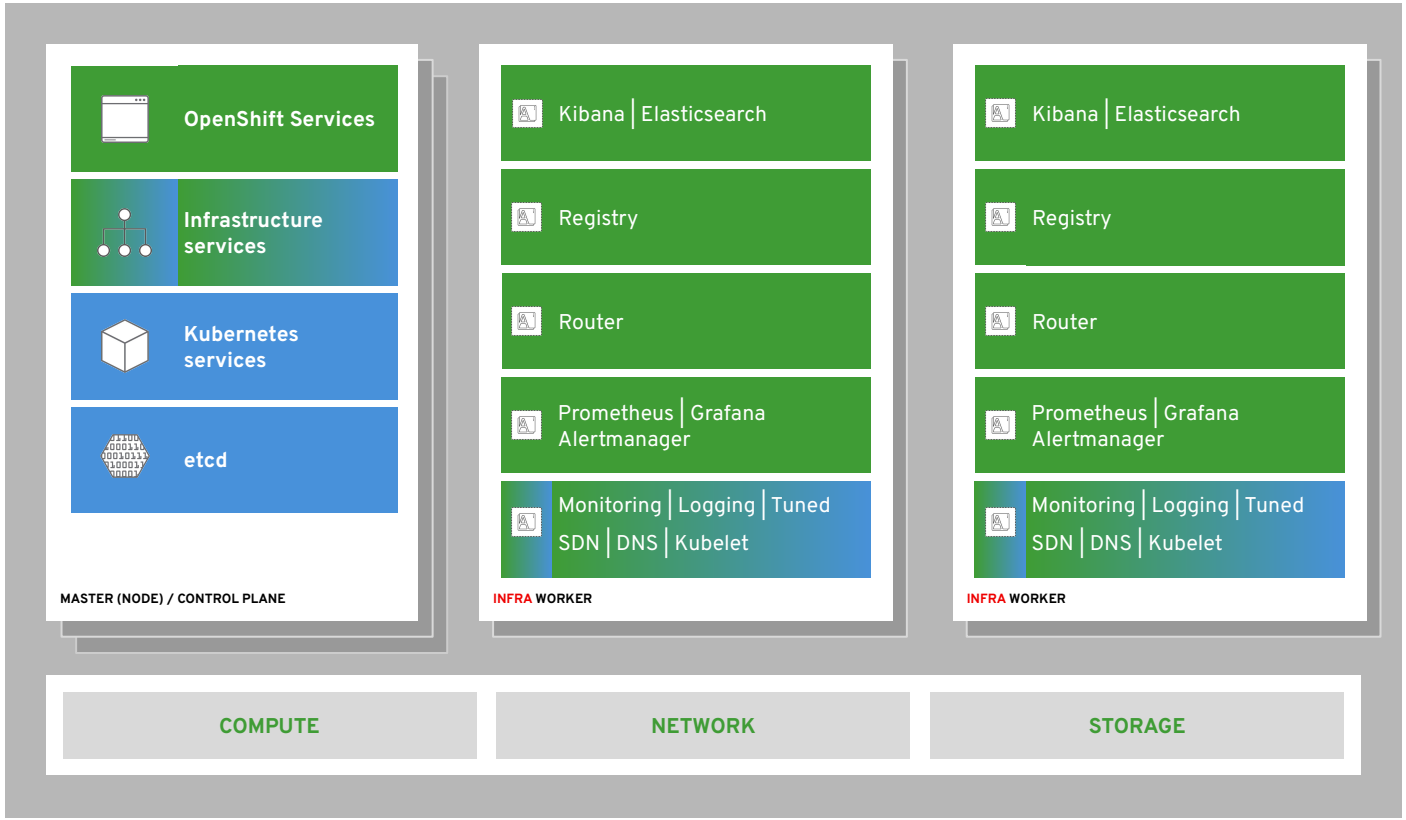
cluster monitoring



log aggregation

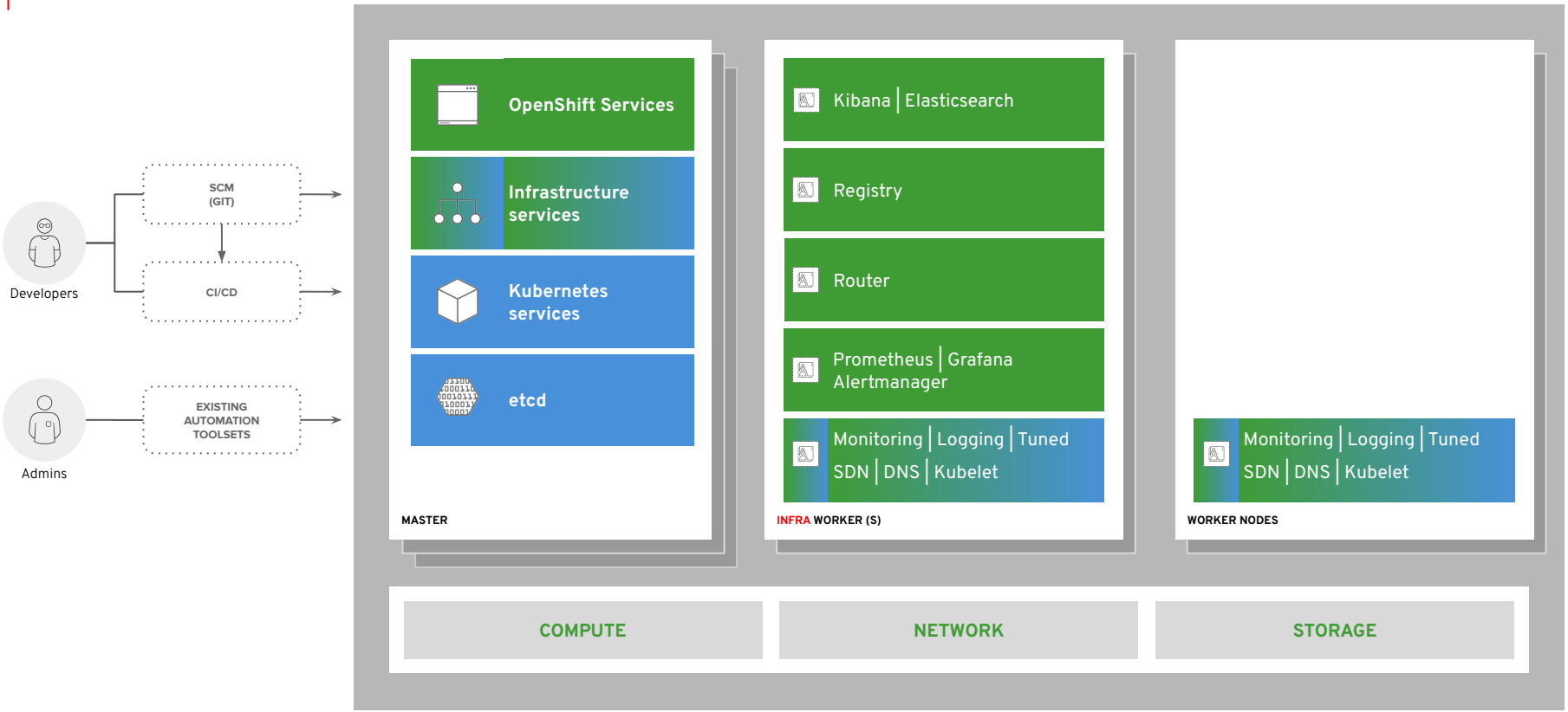


integrated routing



Red Hat

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



Quiz: Describing the Architecture of OpenShift

1. OpenShift is based on which of the following container orchestration technologies?

A ○ Docker Swarm

B ○ Rancher

C ○ Kubernetes

D ○ Mesosphere Marathon

E ○ CoreOS Fleet

CHECK RESET SHOW SOLUTION

2. Which two of the following statements are true of OpenShift Container Platform? (Choose two.)

A ☐ OpenShift provides an OAuth server that authenticates calls to its REST API.

B ☐ OpenShift requires the CRI-O container engine.

C ☐ Kubernetes follows a declarative architecture, but OpenShift follows a more traditional imperative architecture.

D ☐ OpenShift extension APIs run as system services.

CHECK RESET SHOW SOLUTION

3. Which of the following servers runs Kubernetes API components?

A ○ Compute nodes

B ○ Nodes

C ◉ Control plane nodes

CHECK RESET SHOW SOLUTION

4. Which of the following components does OpenShift add to upstream Kubernetes?

A ○ The etcd database

B ○ A container engine

C ○ A registry server

D ○ A scheduler

E ○ The Kubelet

CHECK RESET SHOW SOLUTION

Red Hat

Networking

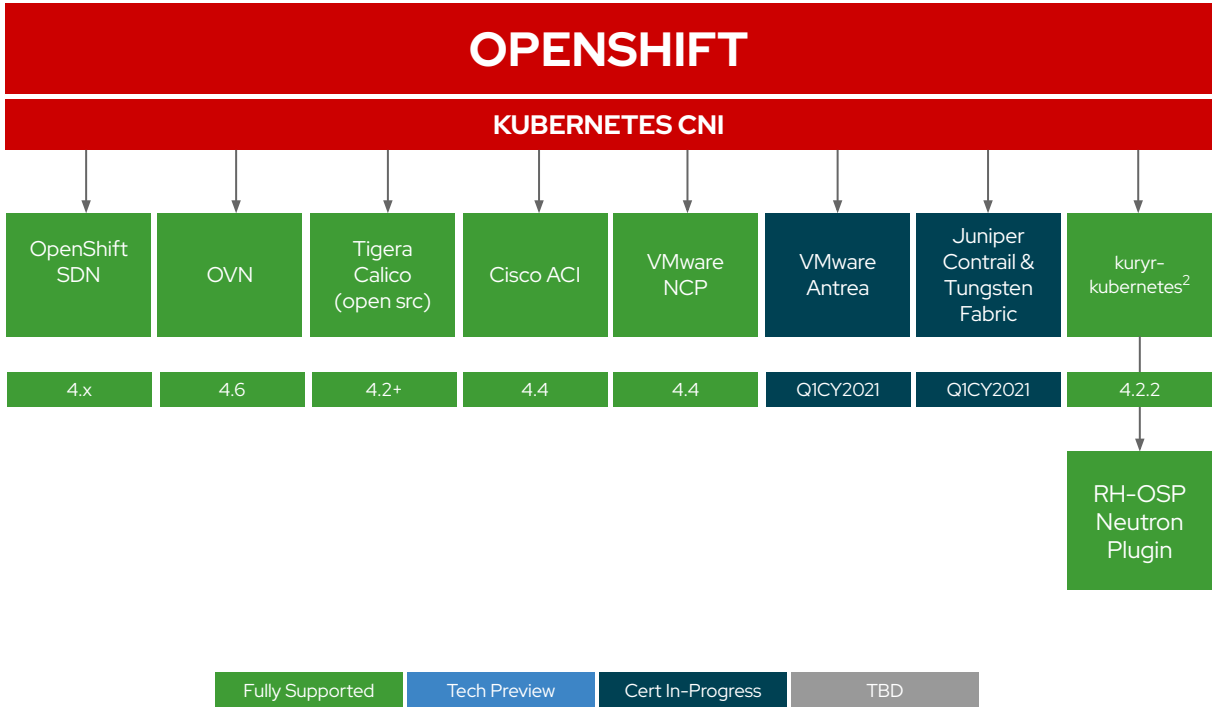
A pluggable model for network interface controls in kubernetes

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 Virtual Network OVN Software Defined Network for kubernetes

OpenShift implements a software-defined network (SDN) to manage the network infrastructure of the cluster and user applications. Software-defined networking is a networking model that allows you to manage network services through the abstraction of several networking layers. It decouples the software that handles the traffic, called the *control plane*, and the underlying mechanisms that route the traffic, called the *data plane*. Among the many features of SDN, open standards enable vendors to propose their solutions, centralized management, dynamic routing, and tenant isolation.

OpenShift SDN high-level architecture

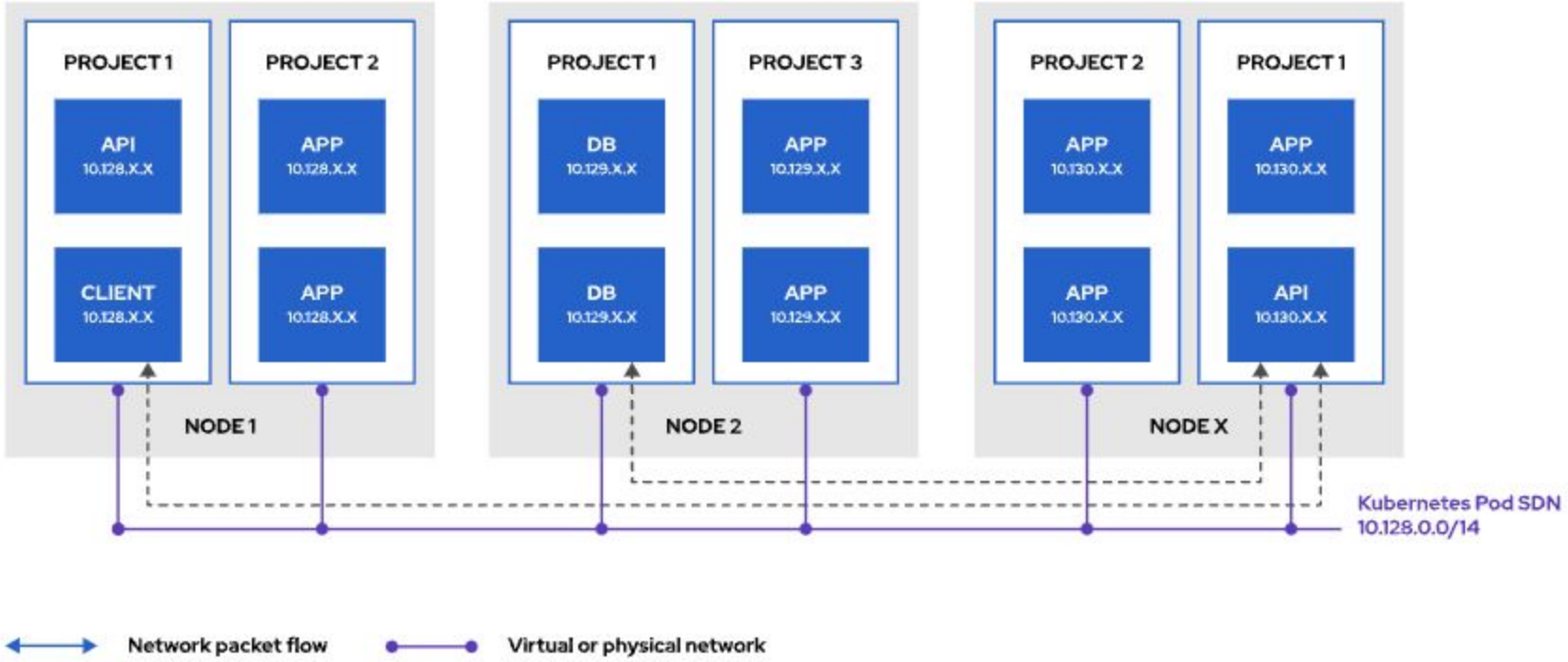


Figure 5.1: Kubernetes basic networking

Using Services for Accessing Pods

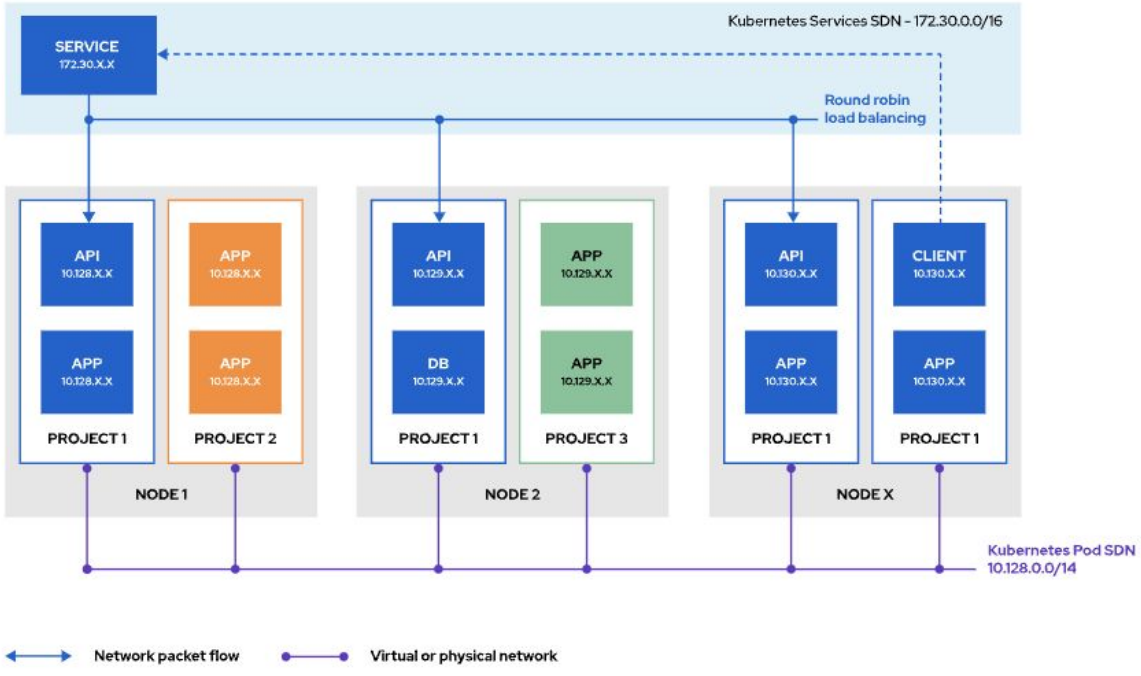
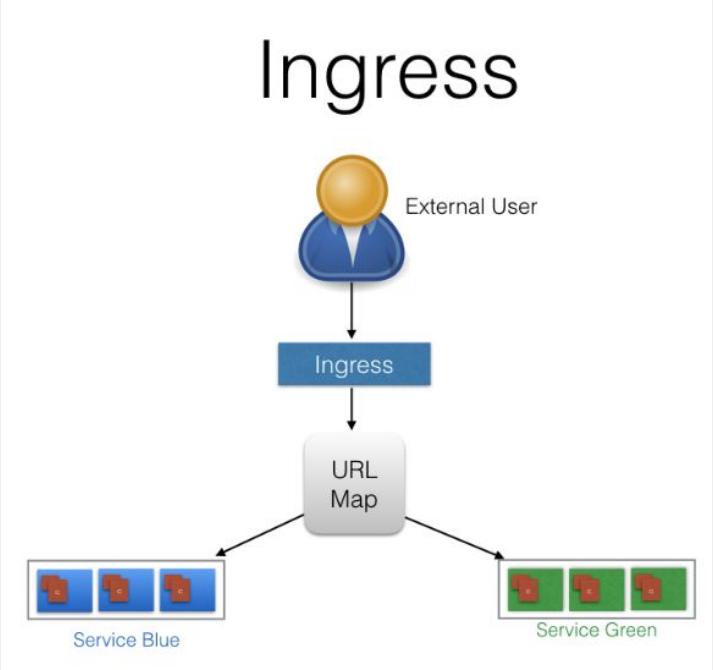


Figure 5.2: Using services for accessing applications

Red Hat

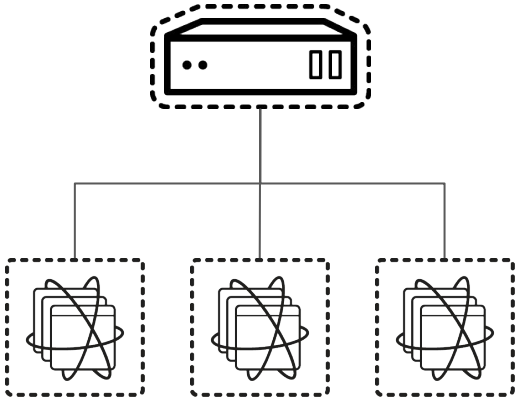
routes and ingress

How traffic enters the cluster



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



Exposing Applications for External Access

Guided Exercise: Exposing Applications for External Access

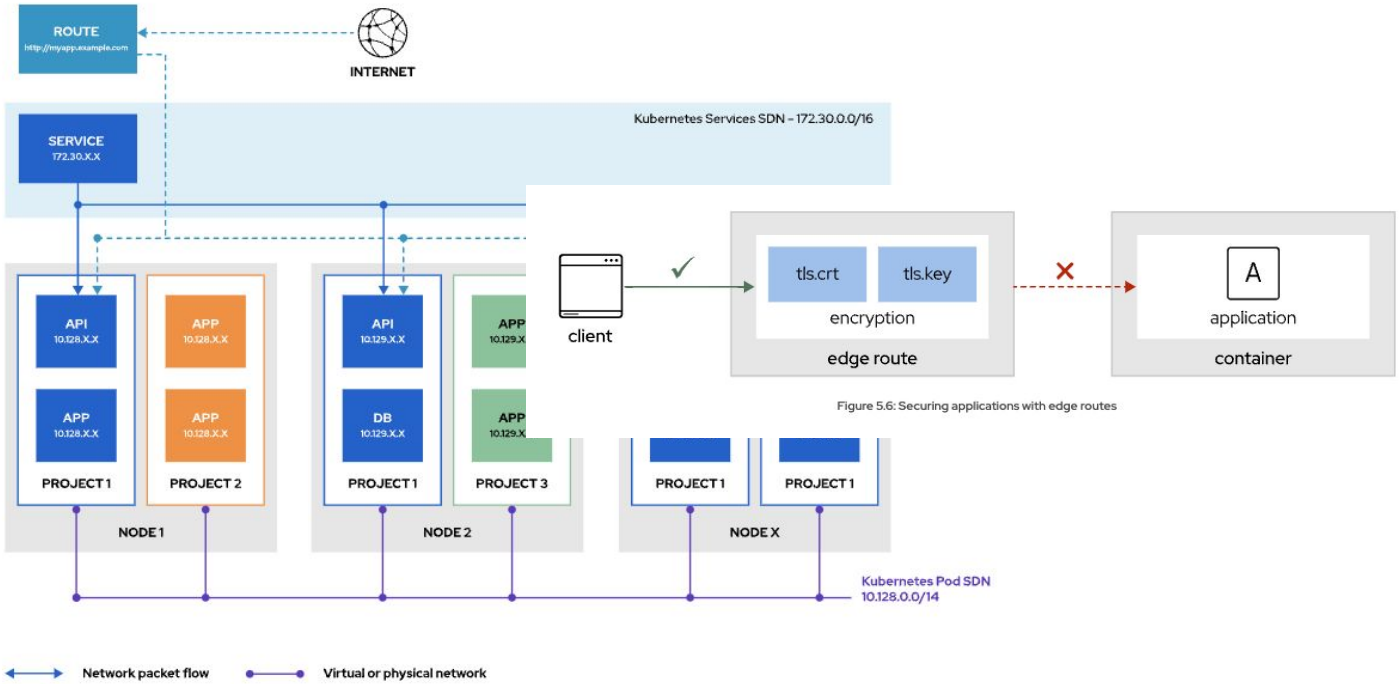


Figure 5.6: Securing applications with edge routes

Figure 5.5: Using routes to expose applications

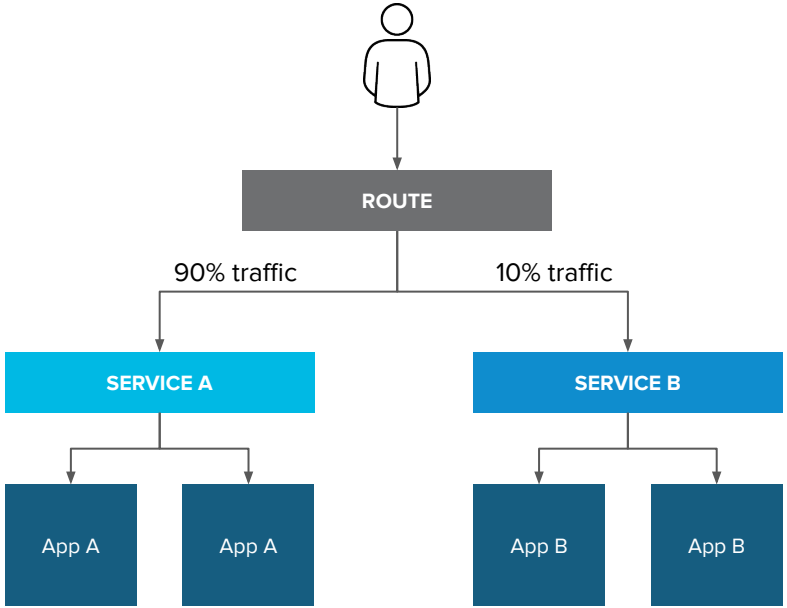
Red Hat

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

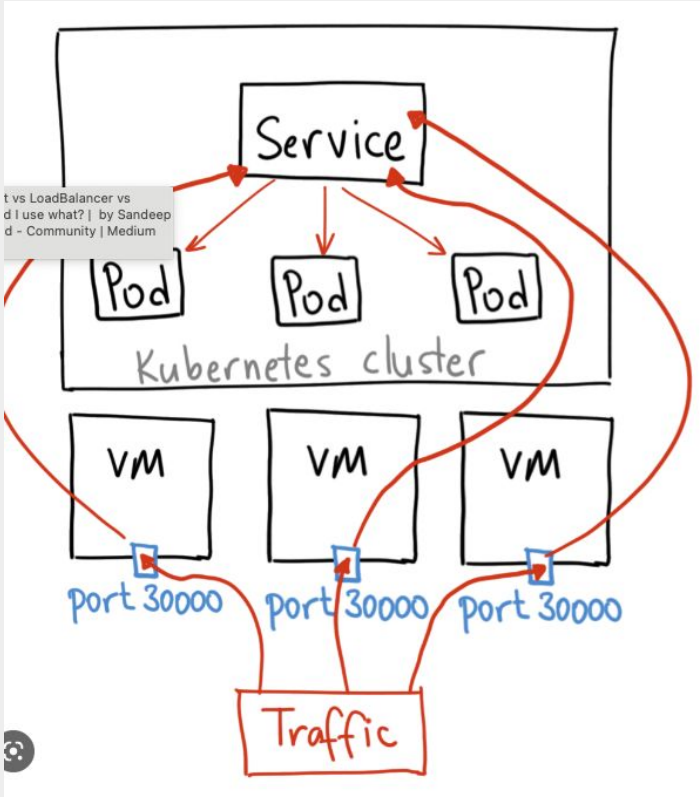
Router-based deployment methodologies

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



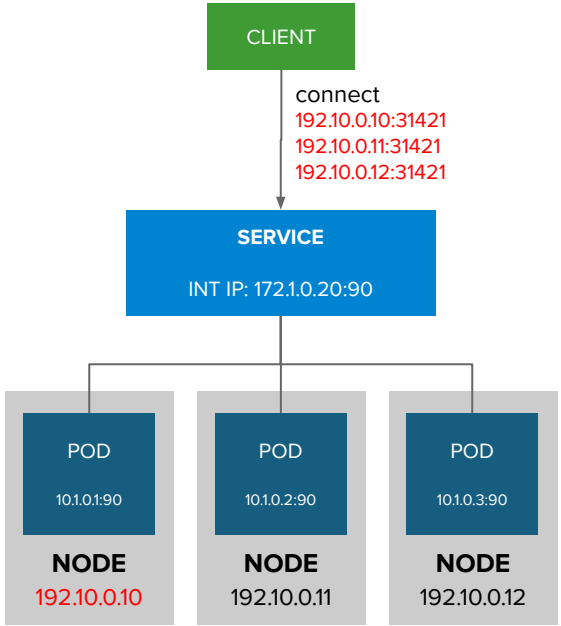
Alternative methods for ingress

Different ways that traffic can enter the cluster without the router



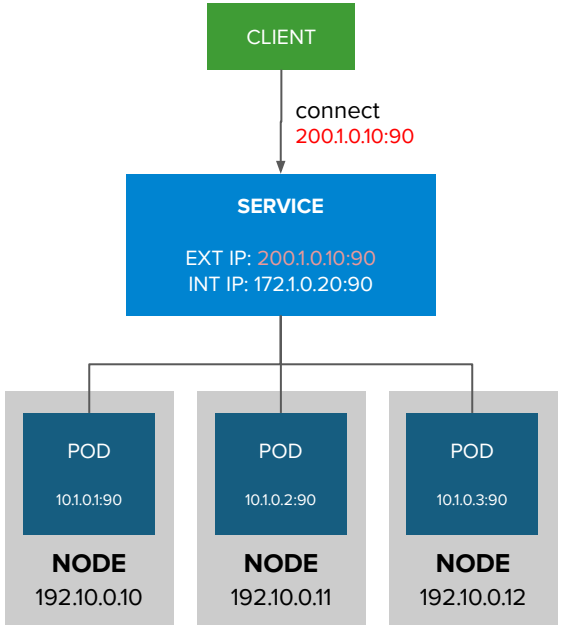
Entering the cluster on a random port with service nodeports

- 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)



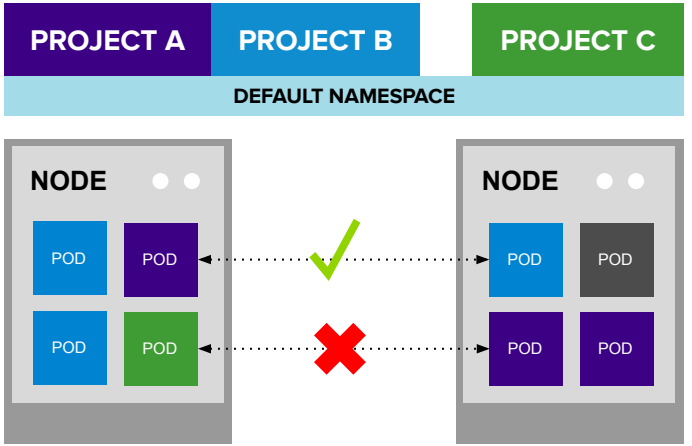
OpenShift SDN “flavors”

OPEN NETWORK **(Default)**

- All pods can communicate with each other across projects

MULTI-TENANT NETWORK

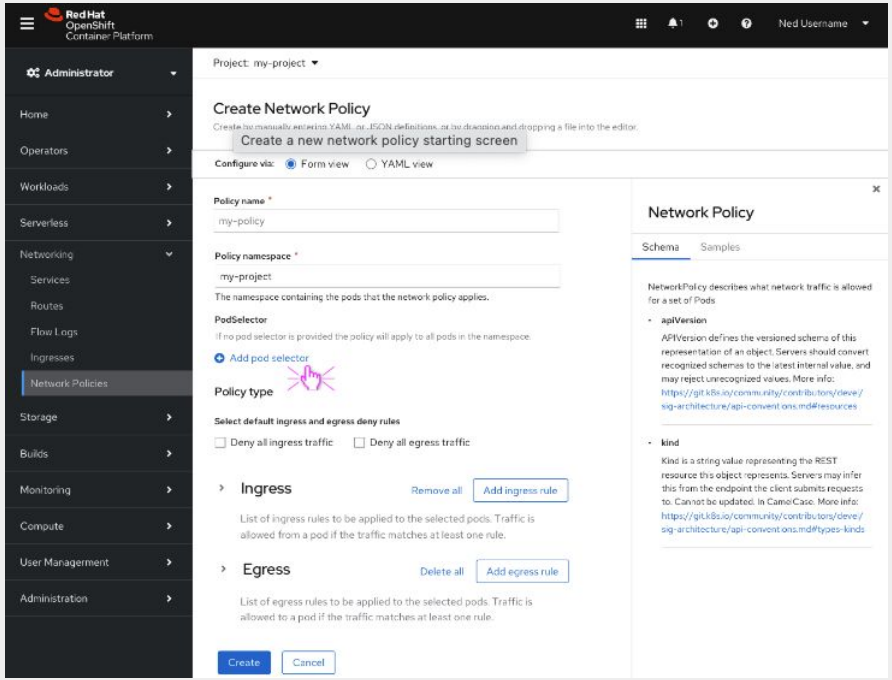
- Project-level network isolation
- Multicast support
- Egress network policies



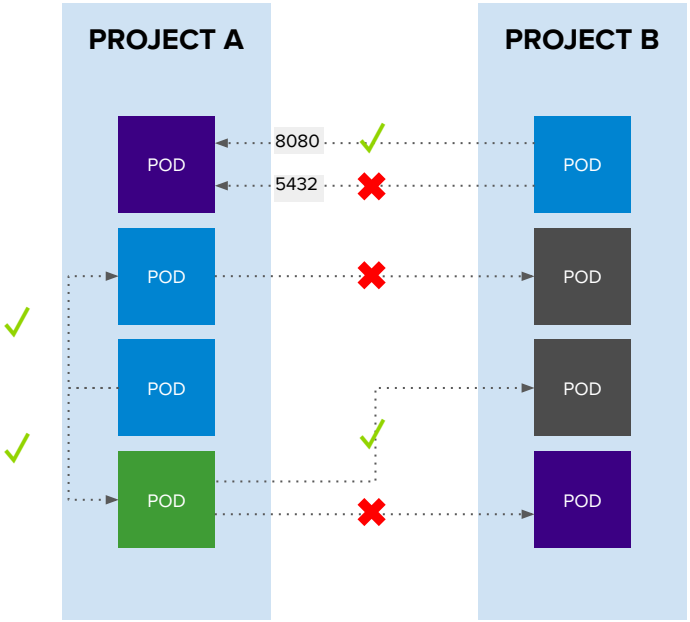
Multi-Tenant Network

Managing Network Policies in OpenShift

Network policies allow you to configure isolation policies for individual pods. Network policies do not require administrative privileges, giving developers more control over the applications in their projects. You can use network policies to create logical zones in the SDN that map to your organization network zones. The benefit of this approach is that the location of running pods becomes irrelevant because network policies allow you to segregate traffic regardless of where it originates.



NetworkPolicy



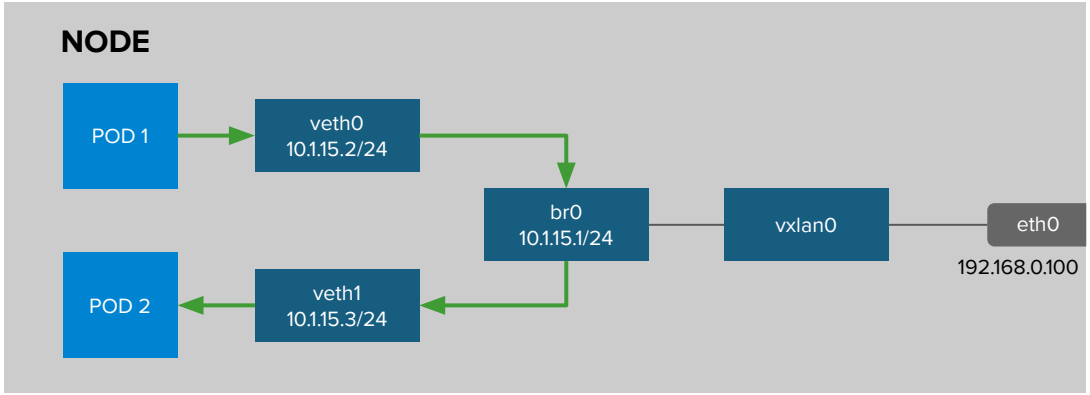
Example Policies

- 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
```

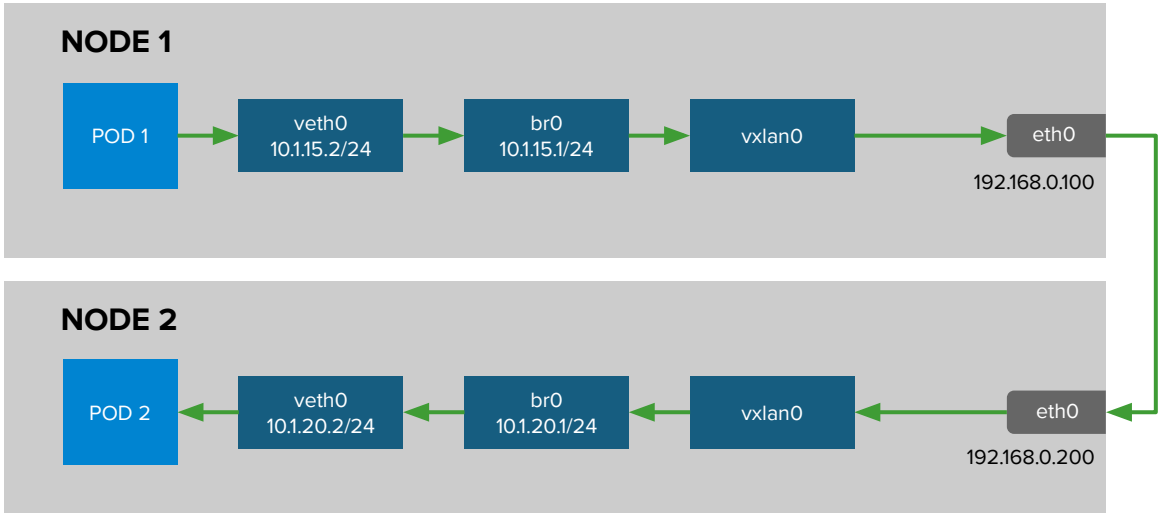
OpenShift SDN packet flows

container-container on same host



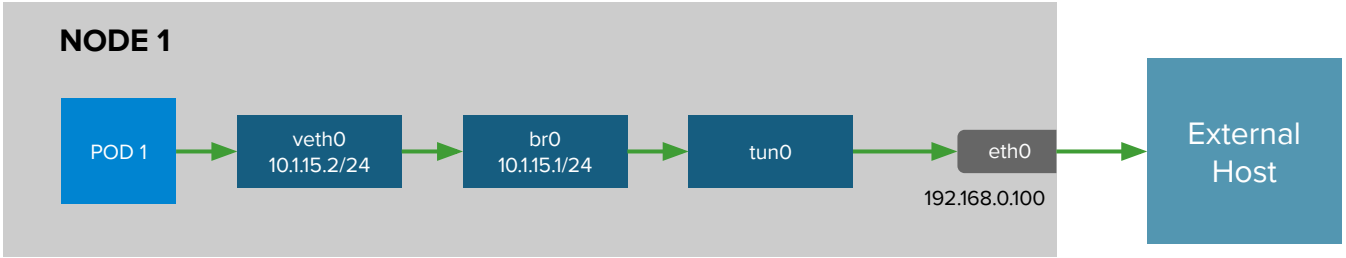
OpenShift SDN packet flows

container-container across hosts



OpenShift SDN packet flows

container leaving the host



Cluster DNS

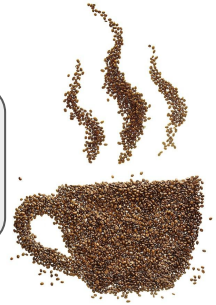
An automated system for providing hostname resolution within kubernetes

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

OpenShift Architecture

Part 1
(16:15 - 17:00)



Part 2
(17:05 - 18:00)

Part 3
Day 2 (16:05 - 17:00)

END of Day 1


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat