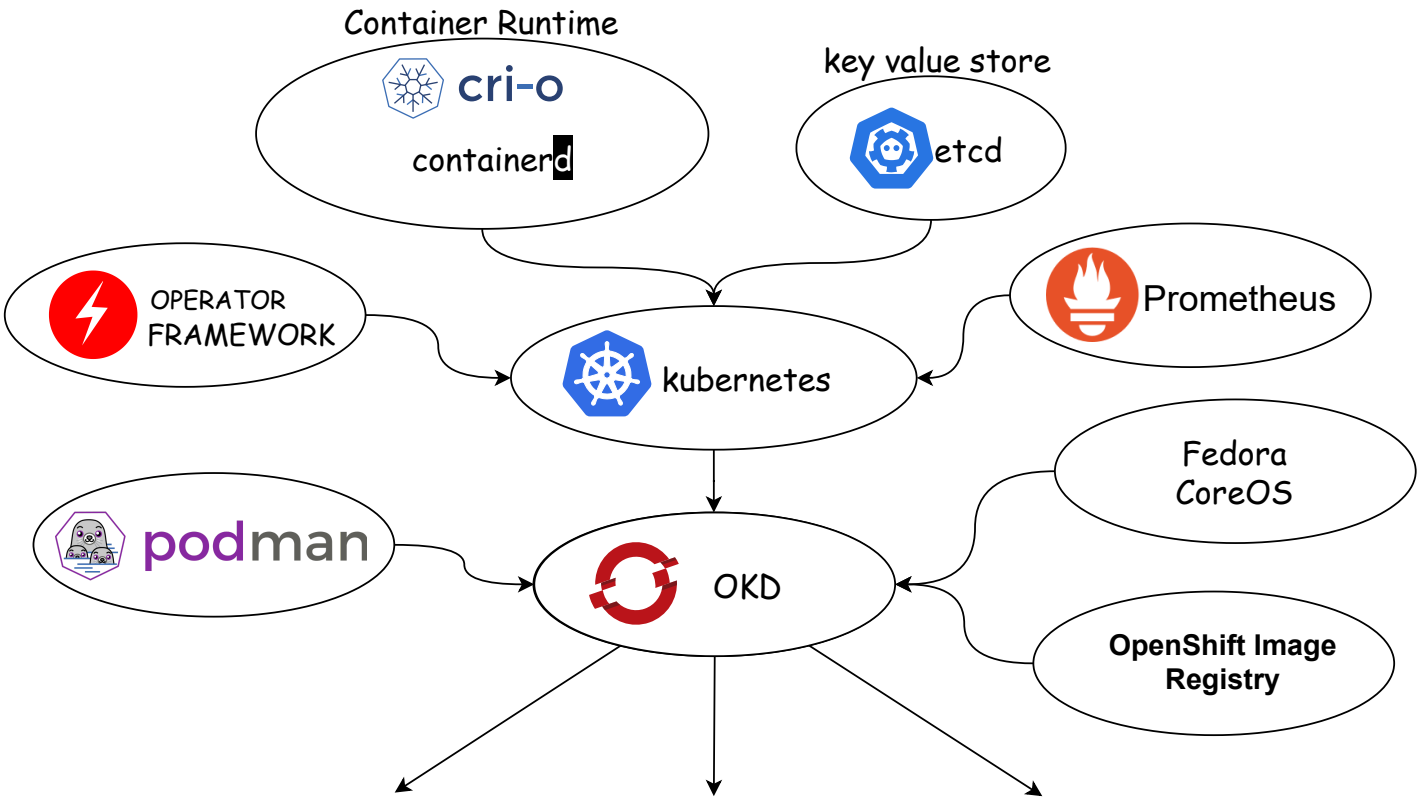


Red Hat OpenShift Container Platform

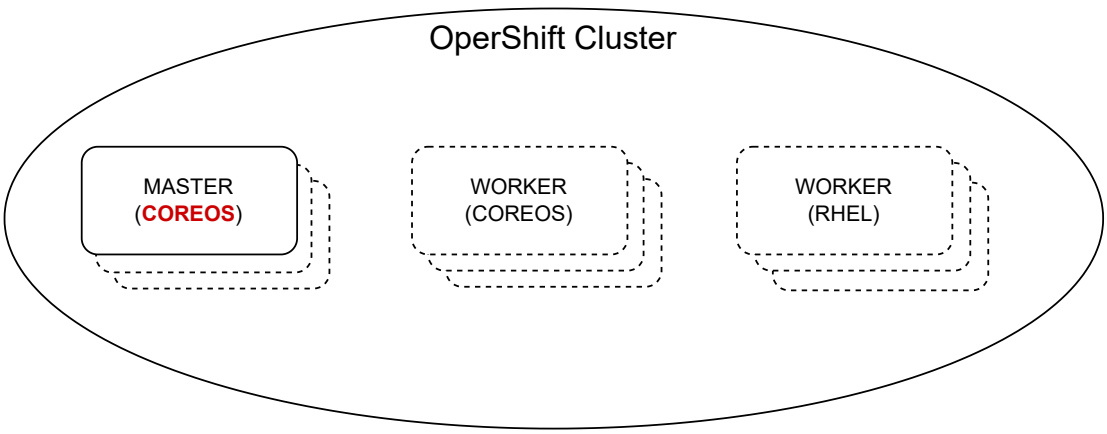
- Public/private DC.
- Bare metal and multiple cloud and virtualization providers.
- Full control by customer.

Red Hat OpenShift Dedicated

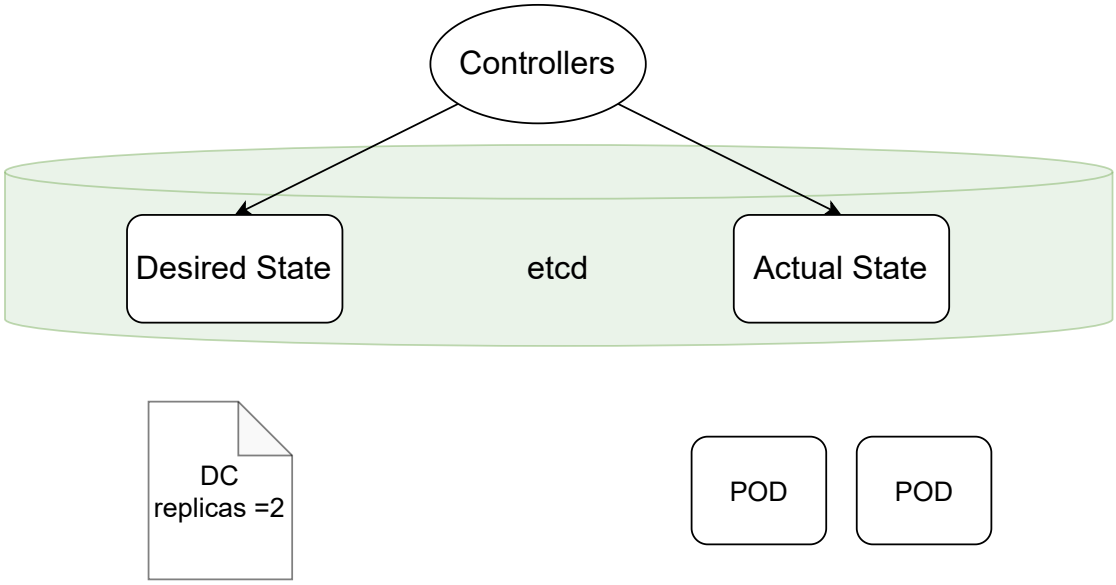
- Managed cluster in public cloud.
- RH manages the cluster.
- Customer manages updates and add-on services.

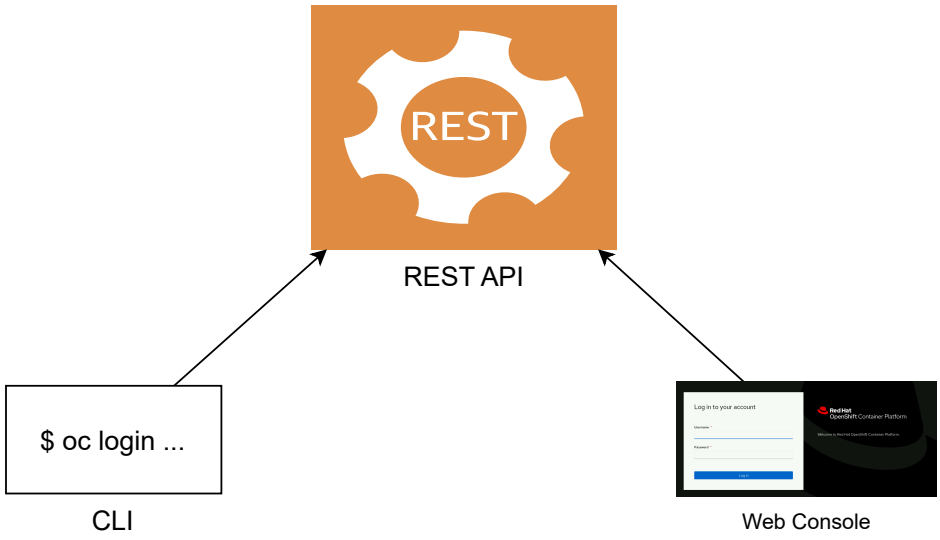
Red Hat OpenShift Online

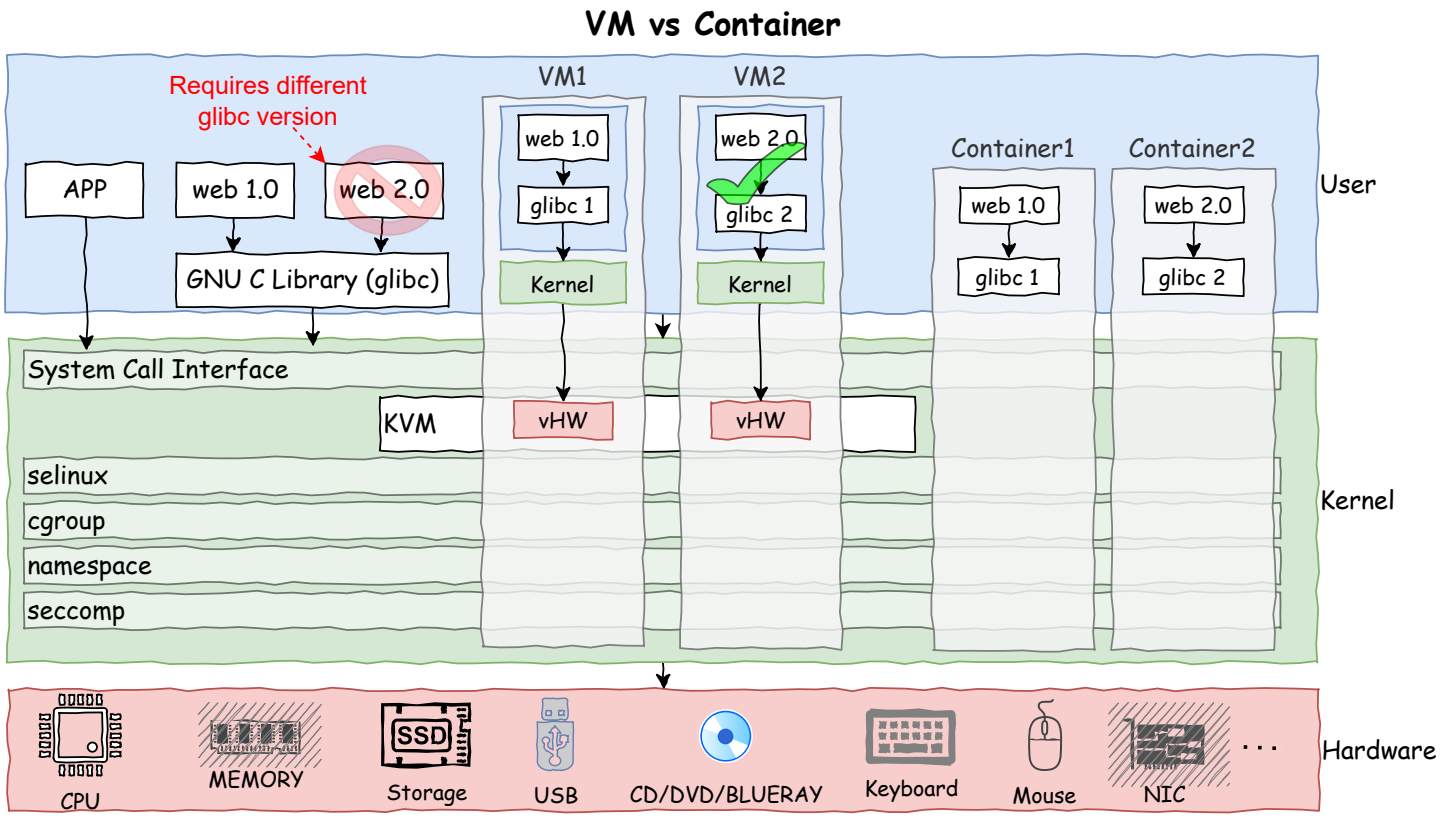
- Public hosted cluster.
- Shared resources by multiple customers.
- RH manages cluster life cycle.

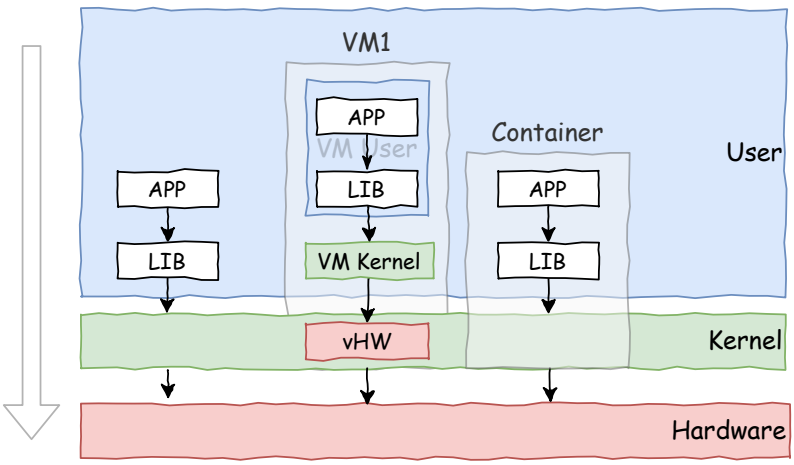


Kubernetes Declarative Architecture

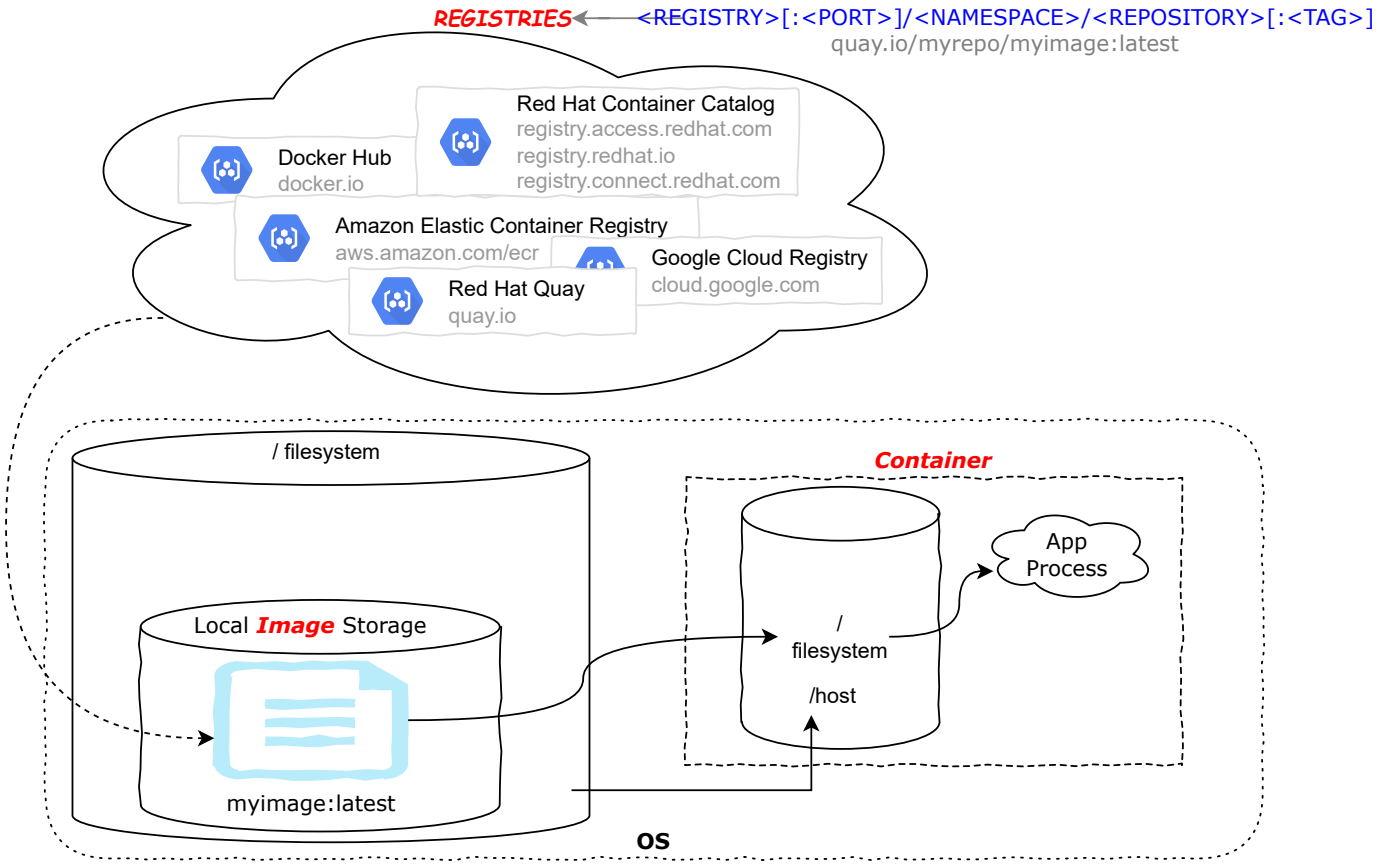




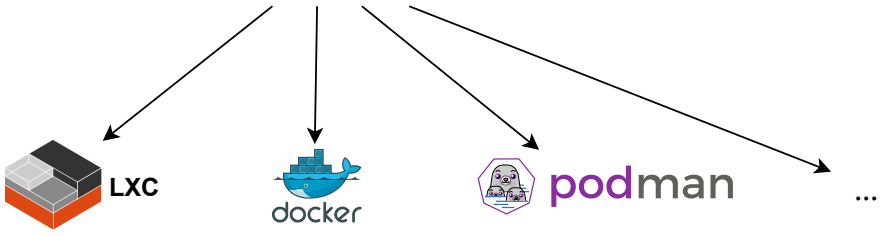

VM vs Container
Requires different glibc version
APP
web 1.0
web 2.0
GNU C Library (glibc)
VM1
web 1.0
glibc 1
Kernel
VM2
web 2.0
glibc 2
Kernel
Container1
web 1.0
glibc 1
Container2
web 2.0
glibc 2
User
System Call Interface
KVM
vHW
vHW
selinux
cgroup
namespace
seccomp
Kernel
CPU
MEMORY
SSD
Storage
USB
CD/DVD/BLUERAY
Keyboard
Mouse
NIC
...
Hardware


VM1
APP
VM User
LIB
VM Kernel
Container
APP
LIB
APP
LIB
User
vHW
Kernel
Hardware

Container Architecture



Container Utilities



- 1st
- OS Containers

- daemon based
- App Containers

- opensource
- ~~daemon based~~
- security focused

OS Container Vs Application Containers

Podman

Image and Registry Operations

podman login [-u *USER*] **[-p** *PASS*] **[***REGISTRY*]	Only if required. Accessing private repo or updating image.
podman logout {-a | *REGISTRY*}	Logout of registry (-a for all).
podman images [-q]	List local images (-q only show id).
podman rmi *IMAGE...*	Removes local image(s). Use -af with caution.
podman search *KEYWORD*	Search registry for an image.
podman pull *SOURCE*	Pull image from a registry.

Where,

SOURCE	**[***REGISTRY*[**:***PORT*]**/***NAMESPACE***/]***IMAGE*[**:***TAG*]
	dir:*PATH*
	docker-archive:*PATH*
	oci-archive:*PATH*

podman tag *IMAGE*[**:***TAG*] *TARGET_NAME*[**:***TAG*]	Add an additional name to a local image
podman push ***IMAGE***	Upload an image to the registry

Container Operations

podman run [--name *NAME*] **[-p** *PORT_INFO*] **[-v** *VOL_INFO*] **[-d] [-it]** *IMAGE* **[***CMD_INFO*]

Where,

--name *NAME*	Container name. Autogenerated if not provided.
-p *PORT_INFO*	**[** *LOCAL_IP* **:]** *LOCAL_PORT* **[[:** *CONT_IP* **] :** *CONT_PORT* **]** Mapping between local IP:PORT to container IP:PORT
-v *VOL_INFO*	*LOCAL_DIR* **:** *CONT_DIR* Mapping between local dir to container dir.
-d	Run in detached mode (background).
-it	-i keep stdin open, -t allocate a pseudo-tty.
IMAGE	Image used to create the container.
CMD_INFO	*CMD* **[***ARG...***]** Command to run in container.

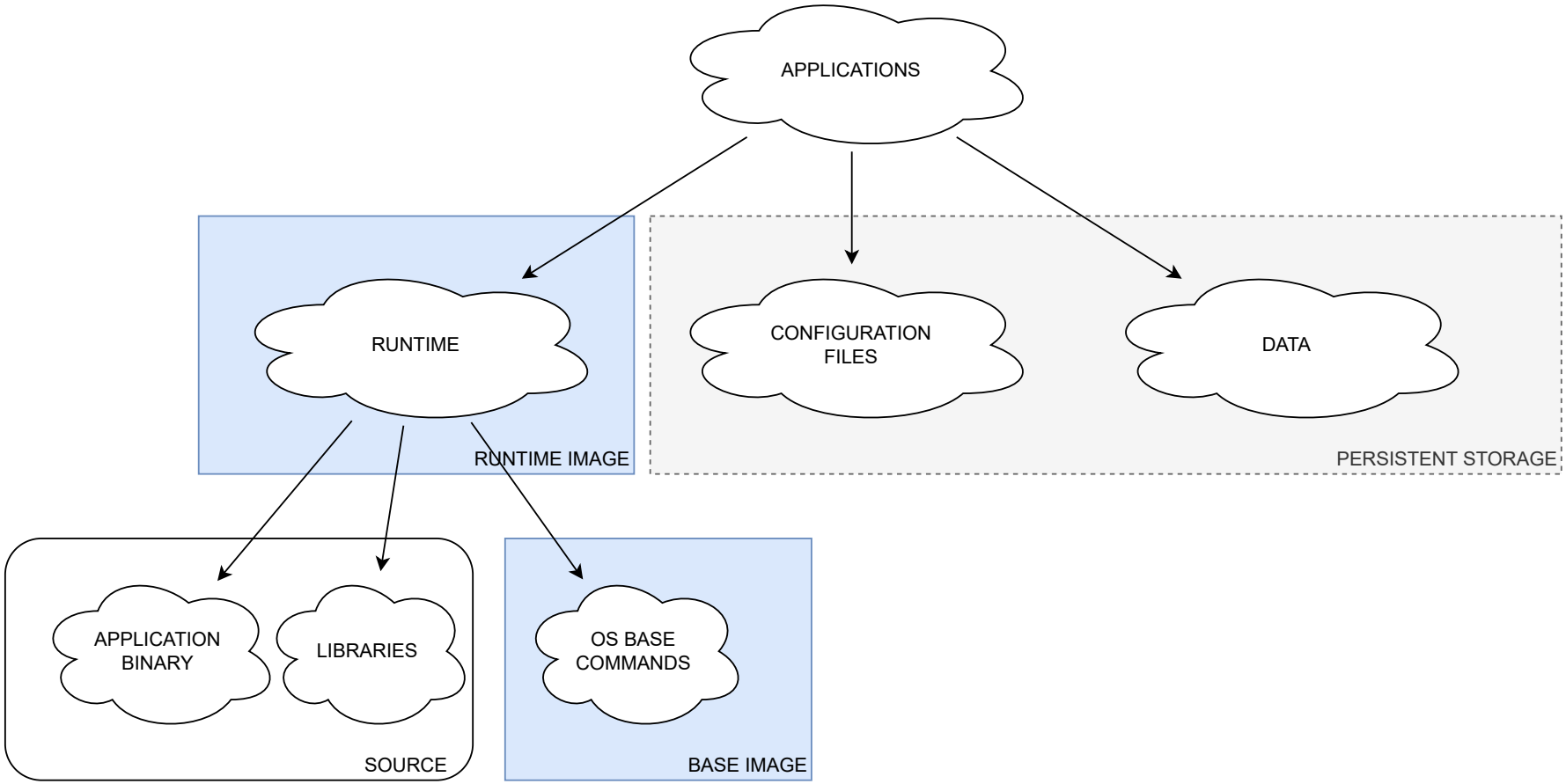
podman exec [-it] *CONTAINER CMD_INFO*	Execute command inside running container.
podman ps [-a] [-q]	List containers (-a for all, -q only show container id).
podman rm *CONTAINER...*	Remove one or more stopped containers. (-f includes running and paused containers).
podman start|stop|restart *CONTAINER...*	Start, stop or restart one or more containers.
podman kill [-s *SIGNAL*] *CONTAINER...*	Send signal to one or more containers.

For more info:

podman --help OR **man podman**

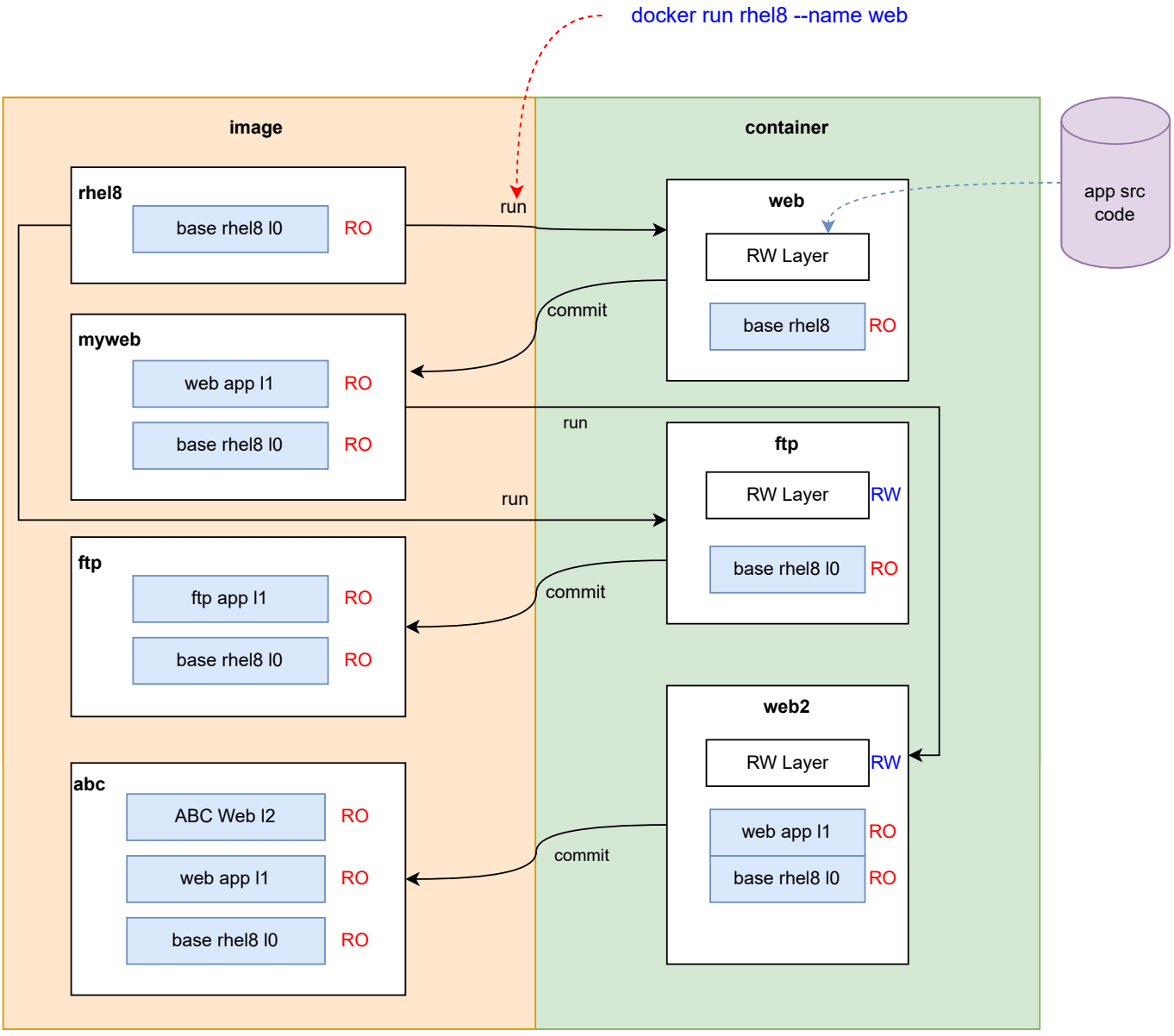
Each sub command has it's own man page. i.e man podman-run, man podman-images, etc.

Basic Container Design

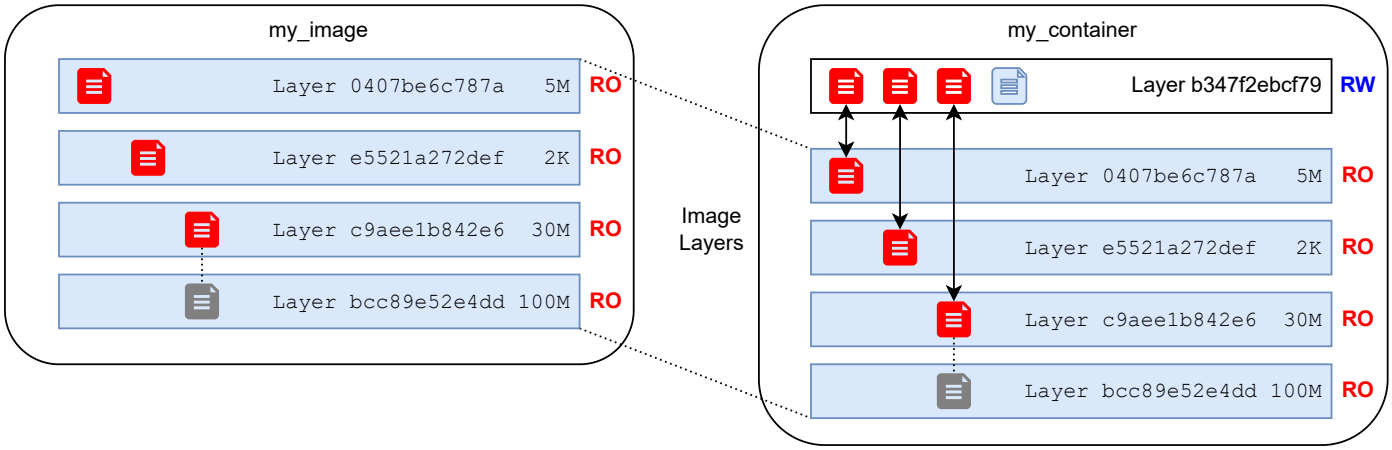


Creating Image

1. Manual
2. Dockerfile/Containerfile
3. Source-To-Image(s2i/STI)
   a) get runtime image and create container
   b) clone source code into container
   c) compile source code
   d) deploy/publish compiled app
   e) cleanup
   f) save container as image

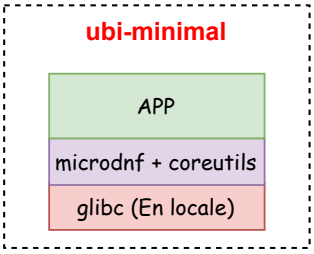


UnionFS - A Stackable Unification File System



BASE IMAGE TYPES

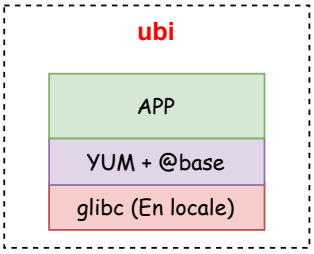
MINIMAL



Designed for apps that contain their own dependencies (Python, Node.js, .NET, etc.)

- Minimized pre-installed content set
- no suid binaries
- minimal pkg mgr (install, update & remove)

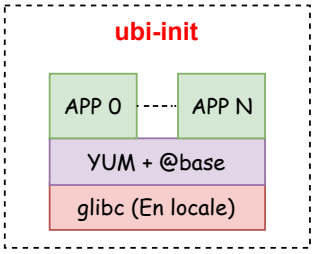
PLATFORM



For any apps that runs on RHEL

- Unified, OpenSSL crypto stack
- Full YUM stack
- Includes useful basic OS tools (tar, gzip, vi, etc)

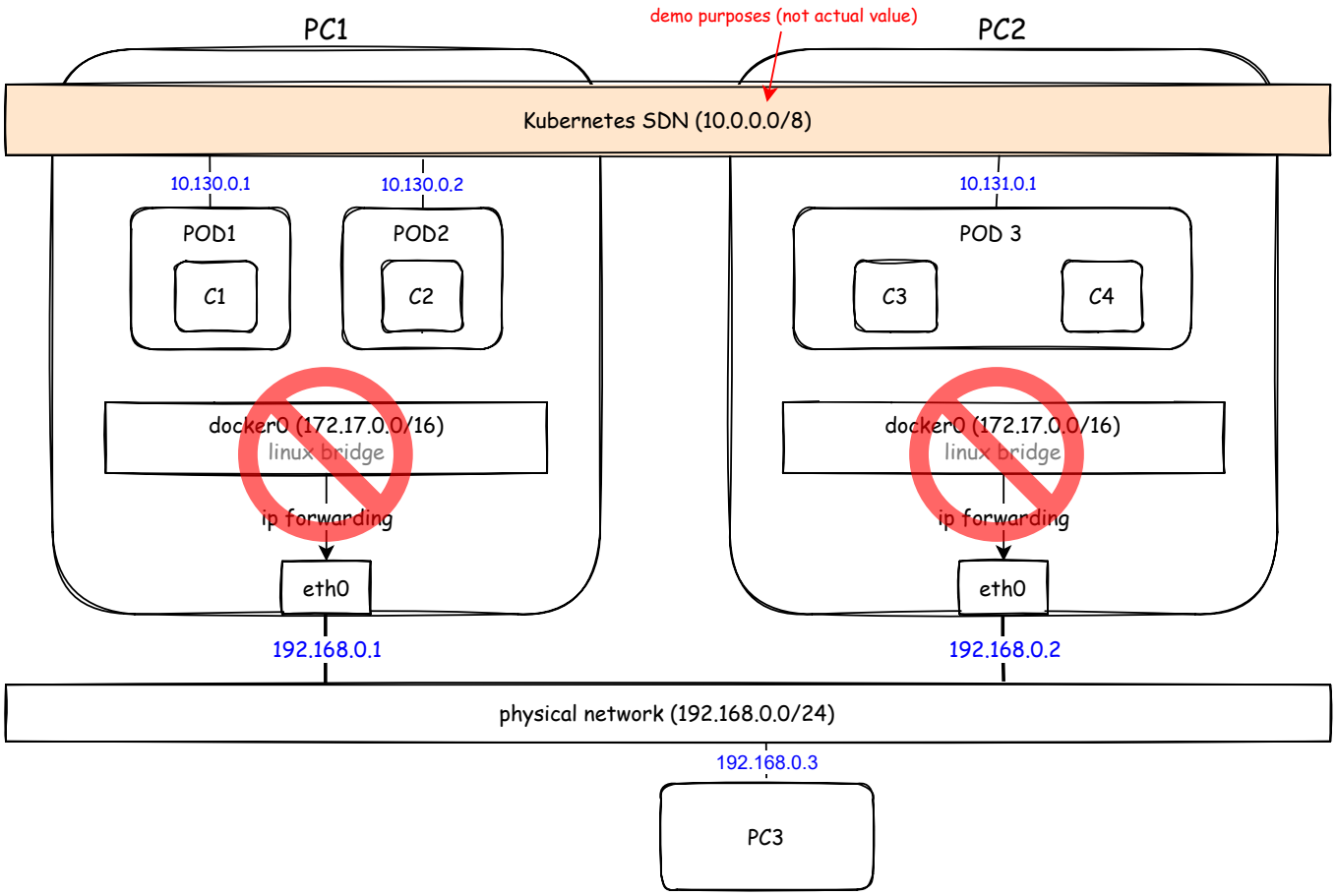
MULTI-SERVICE

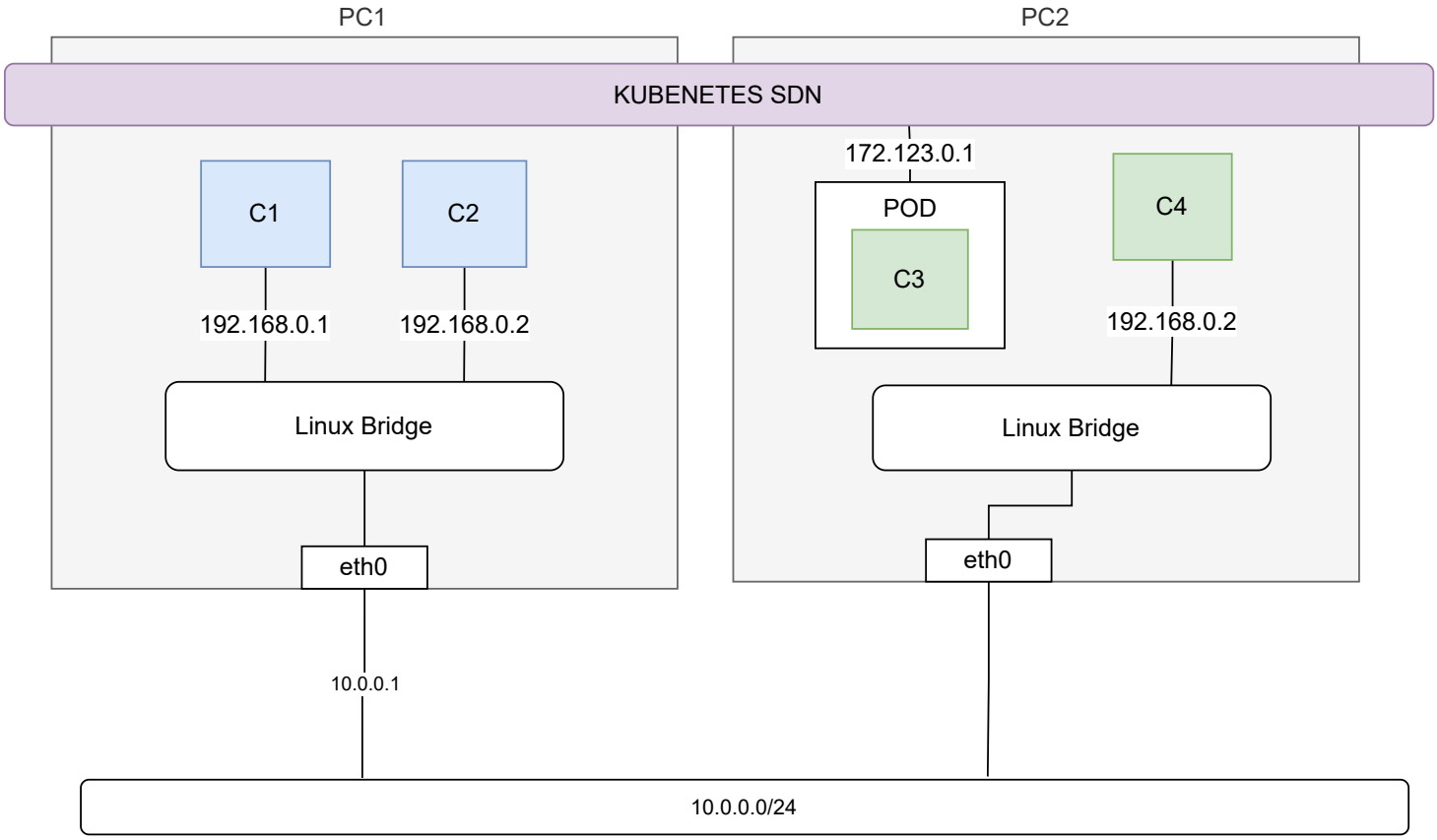


Eases running multi-service in single container

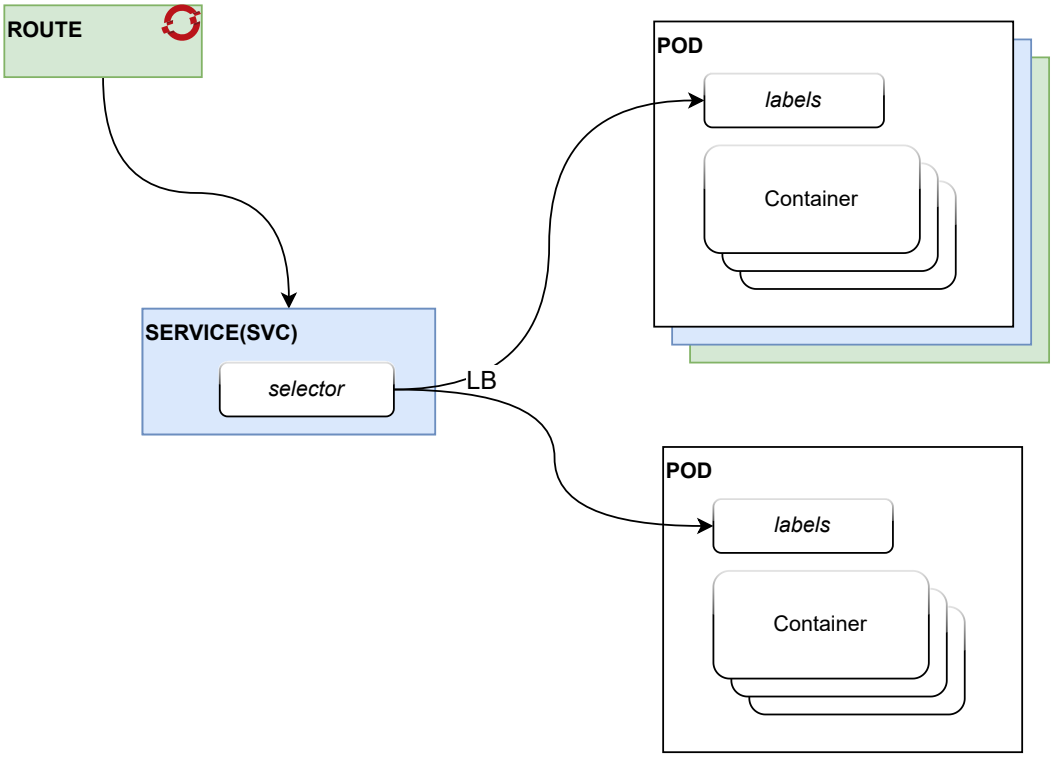
- configured to run systemd on start
- allows you to enable th services at build time

Basic Network - Container vs Kubernetes





Route, Service and Pod Relationship



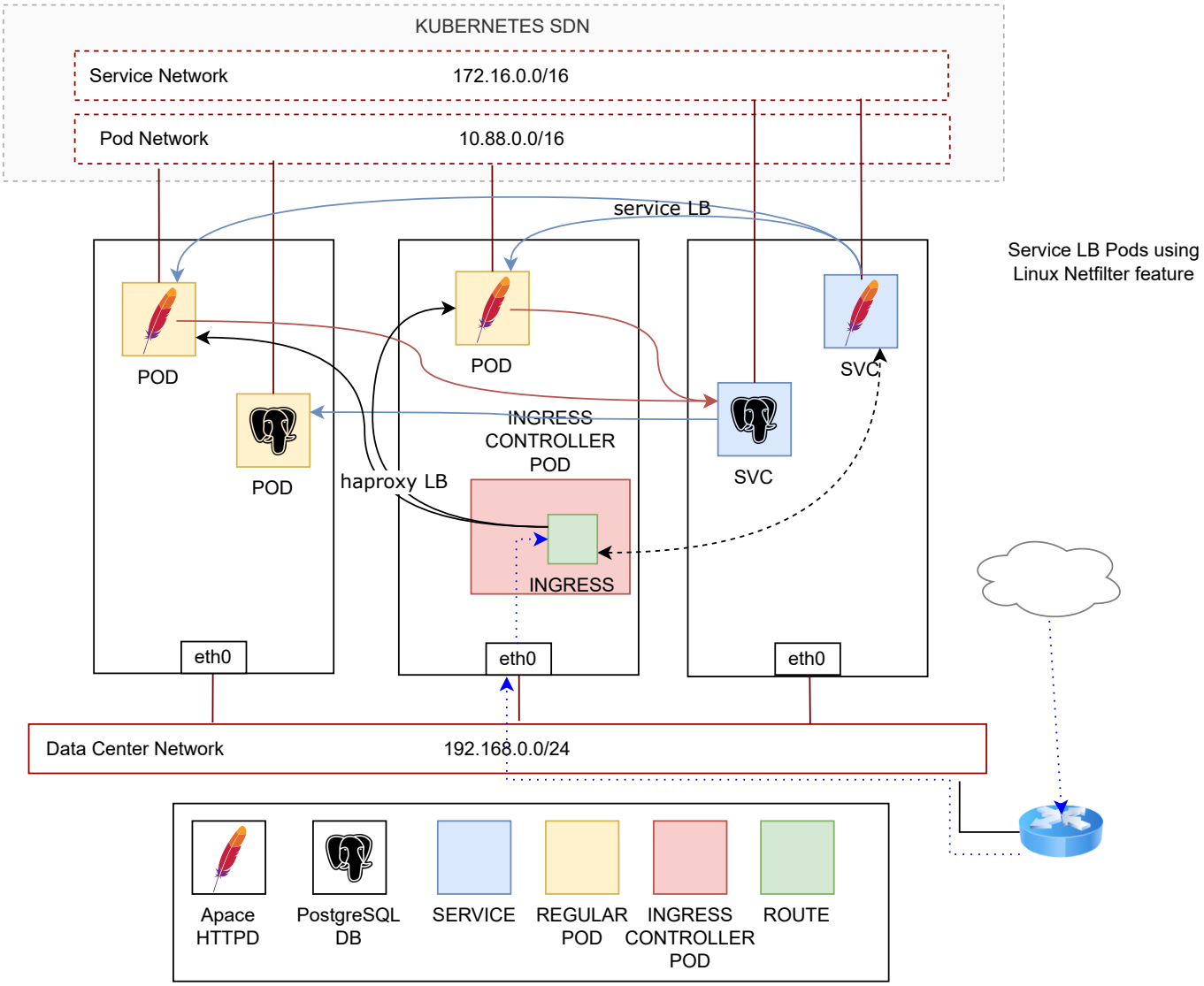
POD
A pod contains one or more containers.

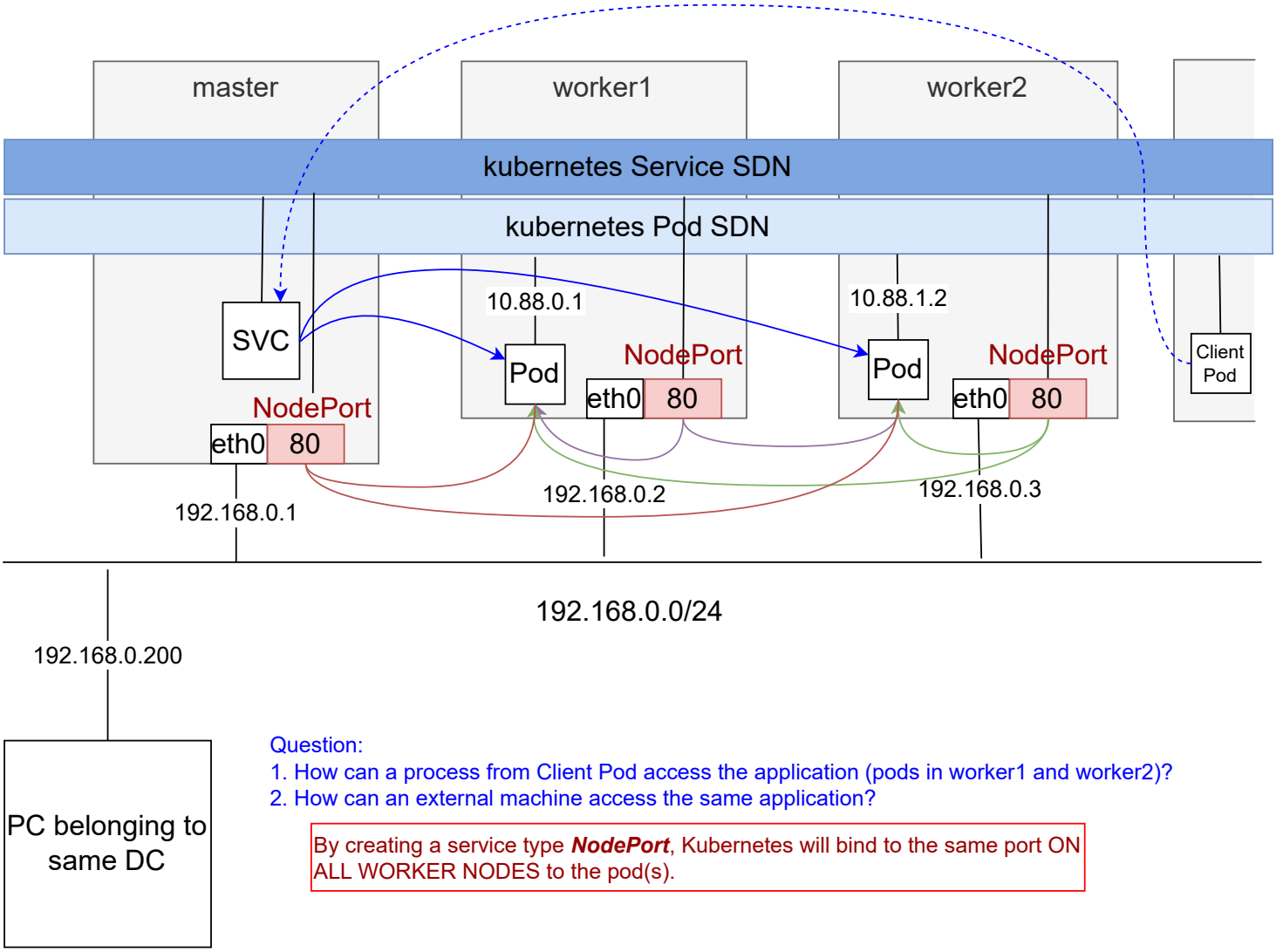
SERVICE
A service references the pod(s) by using the label selector.
The service load balances the connections between all the pods.

ROUTE
A route exposes the service to the external world.

Warning: A service "can" refer to different pods, if the pods have the same label.

Sample of how Services are used

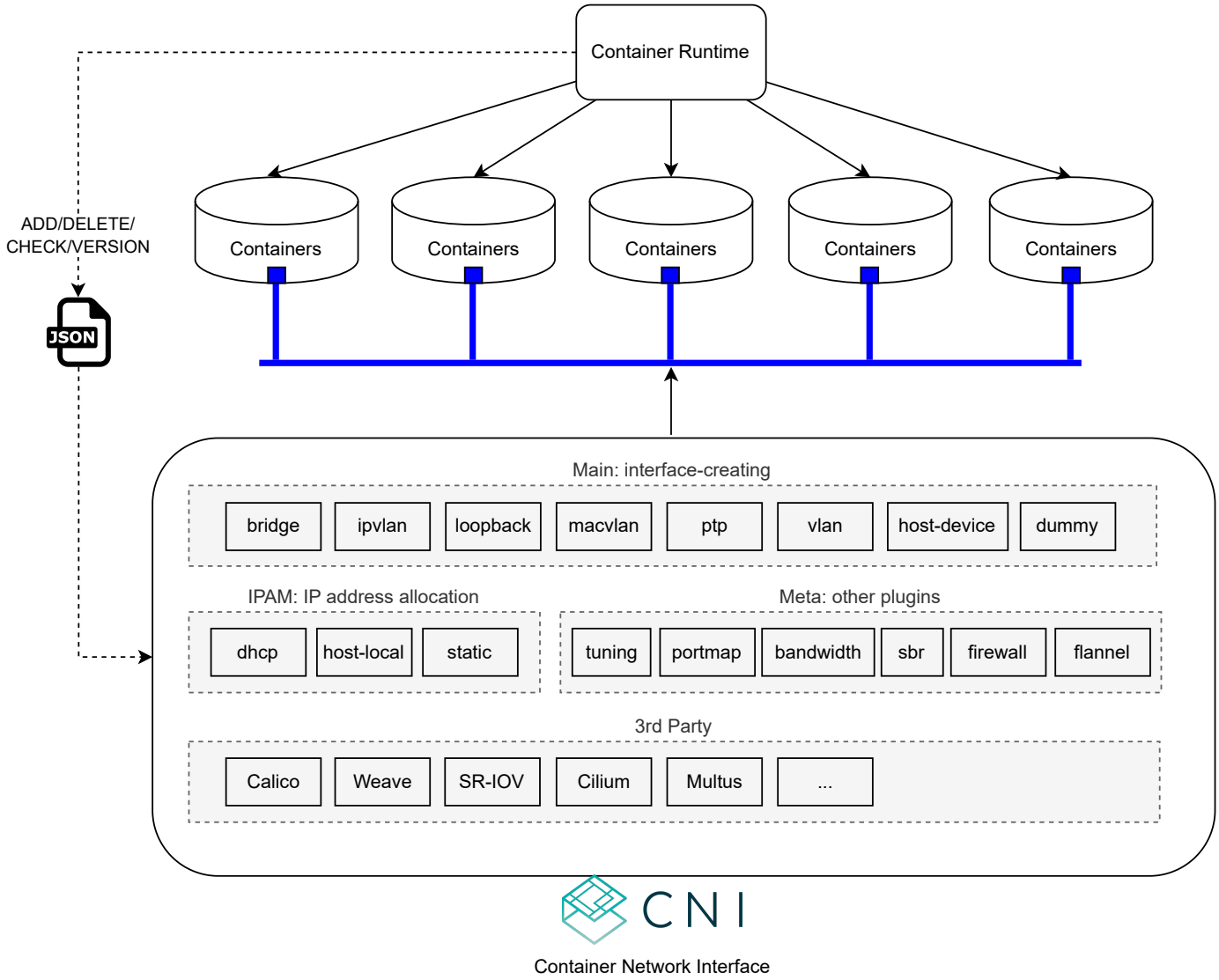




Question:

1. How can a process from Client Pod access the application (pods in worker1 and worker2)?
2. How can an external machine access the same application?

By creating a service type ***NodePort***, Kubernetes will bind to the same port ON ALL WORKER NODES to the pod(s).

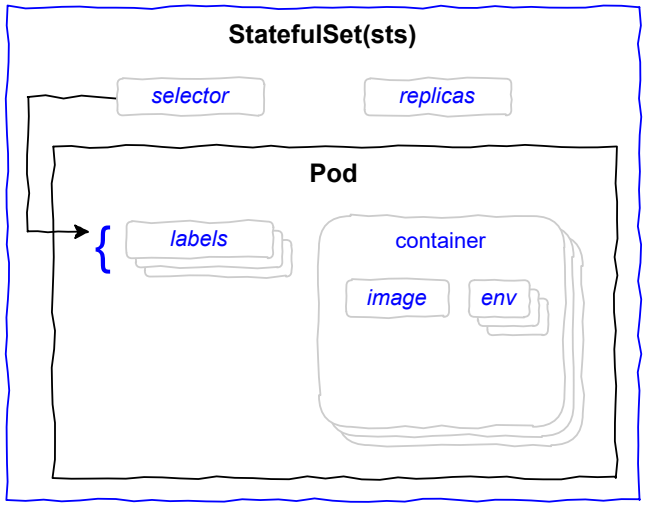


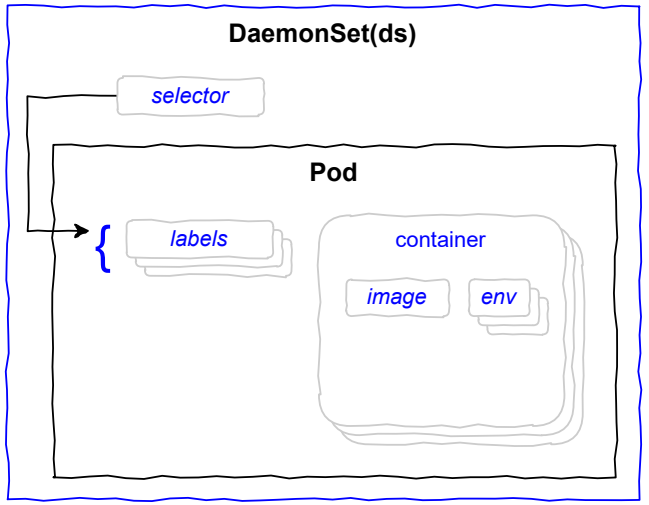
CNI

Container Network Interface

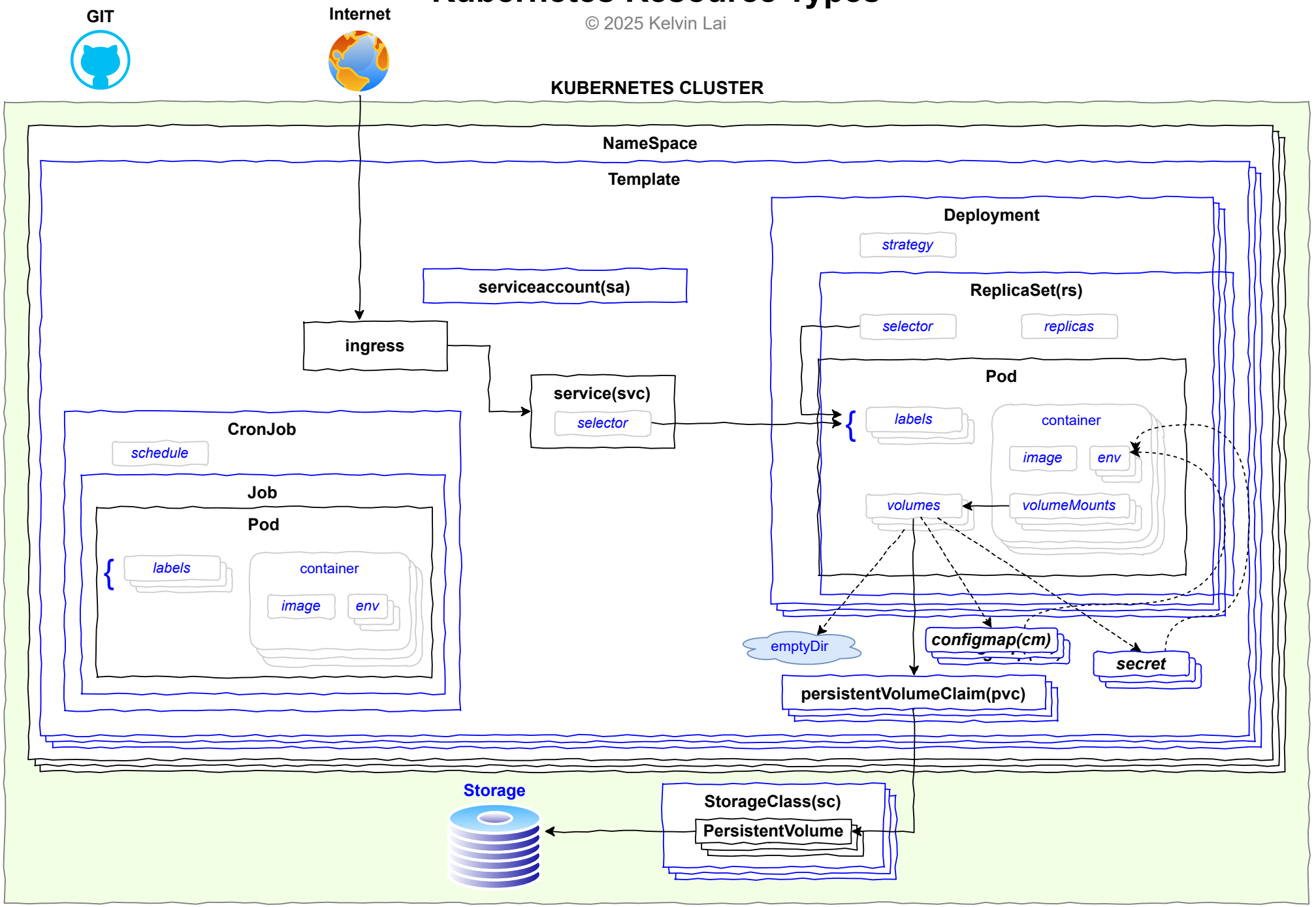
SERVICE(SVC)

```
oc expose <DC/DEPLOYMENT/RC/RS/POD> <RESOURCE_NAME>

    DNS NAME = <SVC>.<PROJ>[.svc.cluster.local]
 ENVIRONMENT VARIABLE IN POD = <SVC>_SERVICE_HOST
```





Kubernetes Resource Types

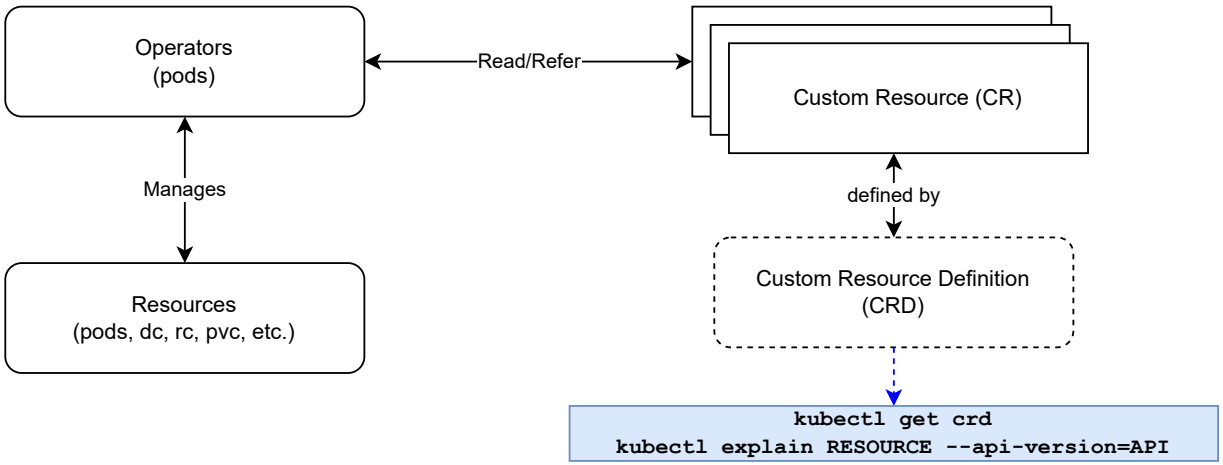


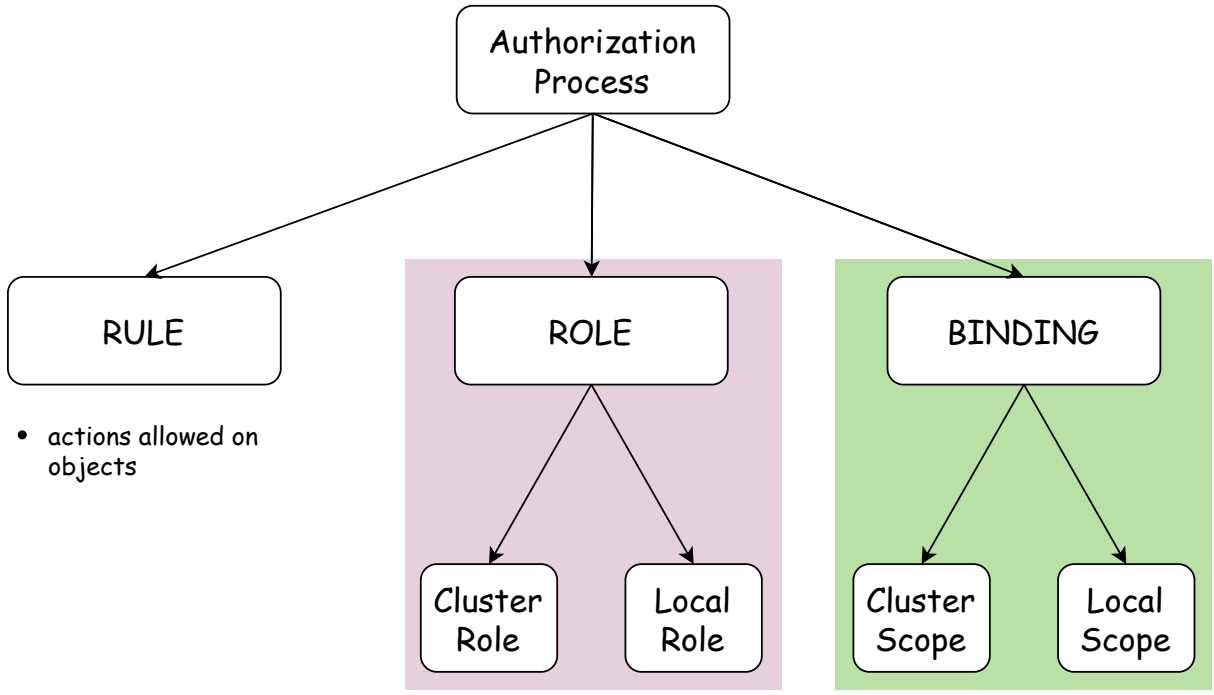
Template

```
apiVersion: template.openshift.io/v1

kind: Template

metadata:
  name: mytemplate
  annotations:
    description: "Description"

objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: ${APP_NAME}
  spec:
    containers:
    - env:
      - name: ACCESS_CODE
        value: ${APP_PASS}
      image: superapp/hyperimage
      name: myApp
      ports:
      - containerPort: 8080
        protocol: TCP

parameters:
- description: Name of Pod
  name: POD_NAME
  value: myPod
  required: true
- description: Application Secret Access Code
  name: APP_PASS
  generate: expression
  from: "[a-zA-Z0-9]{8}"

labels:
  mylabel: myapp
```

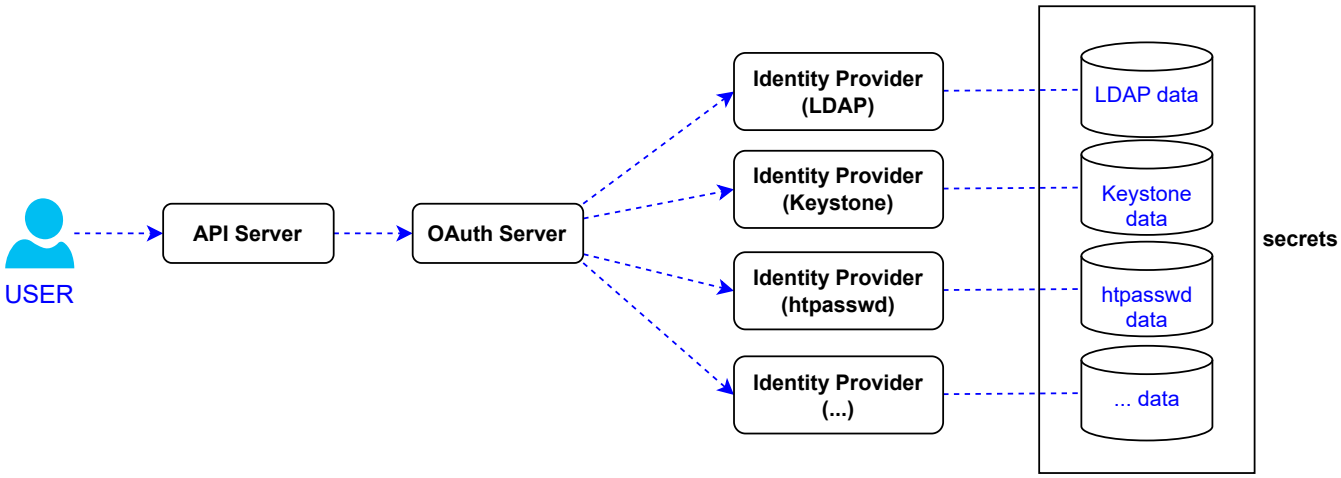
```
pipeline {
  options {                    // Global Options - can be overridden per stage basis
    timeout(time: 30, unit: 'MINUTES')
  }
  agent {                      // Execution Context (where to run)
    node {
      label 'master'           // IS used to run pipeline,  (master - default minimal linux runtime)
    }
  }
  environment {                // Global Vars
    DEV_PROJ = “super-app-v1”
    APP_NAME = “myapp”
  }
  stages {
    stage('stage 1') {
      steps {
        script {            // DSL code (Groovy language code) must be embedded within script element
          openshift.withCluster() {
            openshift.withProject(env.DEV_PROJ) {
              openshift.selector("bc", "${APP_NAME}").startBuild("--wait=true", "--follow=true")
            }
          }
        }
      }
    }
    stage('stage 2') {      // NOTE: project switches to default proj between stages.
      steps {
        sh 'echo hello from stage 2!'
      }
    }
    stage('manual approval') {
      steps {
        timeout(time: 60, unit: 'MINUTES') {
          input message: "Move to stage 3?"
        }
      }
    }
    stage('stage 3') {
      steps {
        sh 'echo hello from stage 3!. This is the last stage...'
      }
    }
  }
}
```

An operator is a control loop program and it can respond to events.
It communicates with the API server to manage k8s resources





- actions allowed on objects




```
$ oc get co | grep auth
$ oc get pods -n openshift-authentication-operator
```

Project: *openshift-authentication-operator*

Authentication Operator

Read

OAuth CR

Manages/Configures

OAuth Server

Issues Access Tokens

Project: *openshift-authentication*

```
$ oc get pods -n openshift-authentication
```

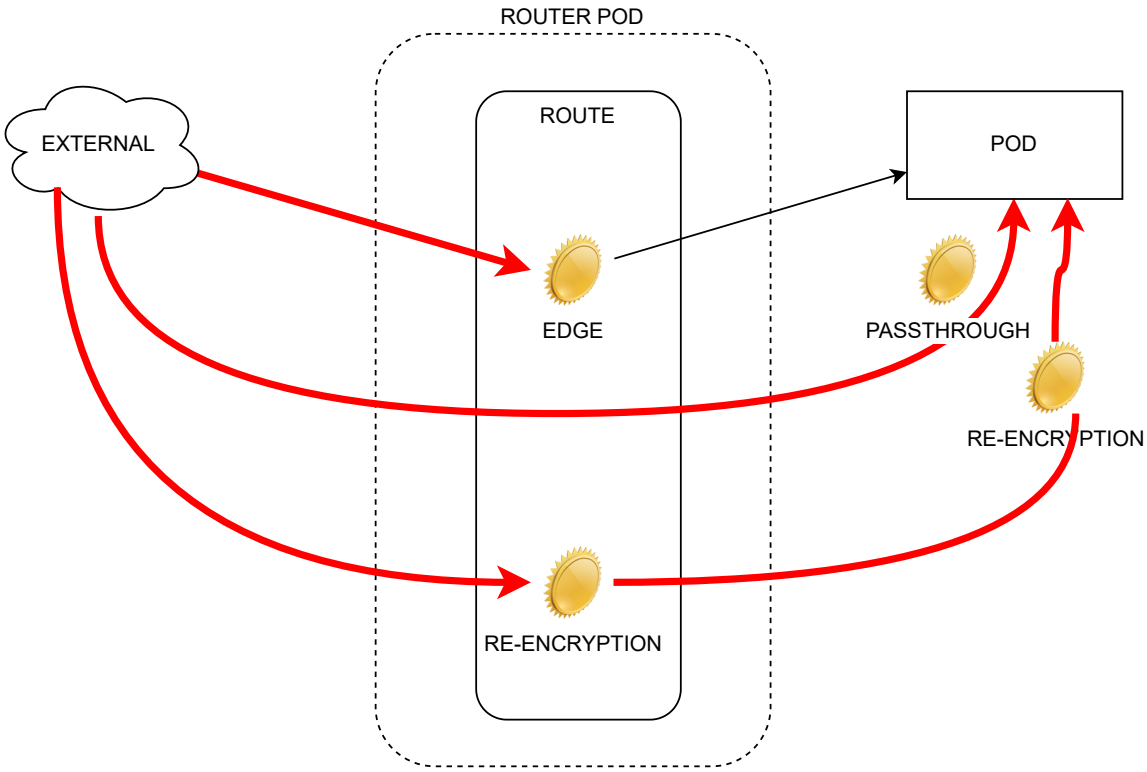
```
$ oc get oauth cluster -o yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
  ...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: htpasswd-secret
    mappingMethod: claim
    name: htpasswd_provider
    type: HTPasswd
```

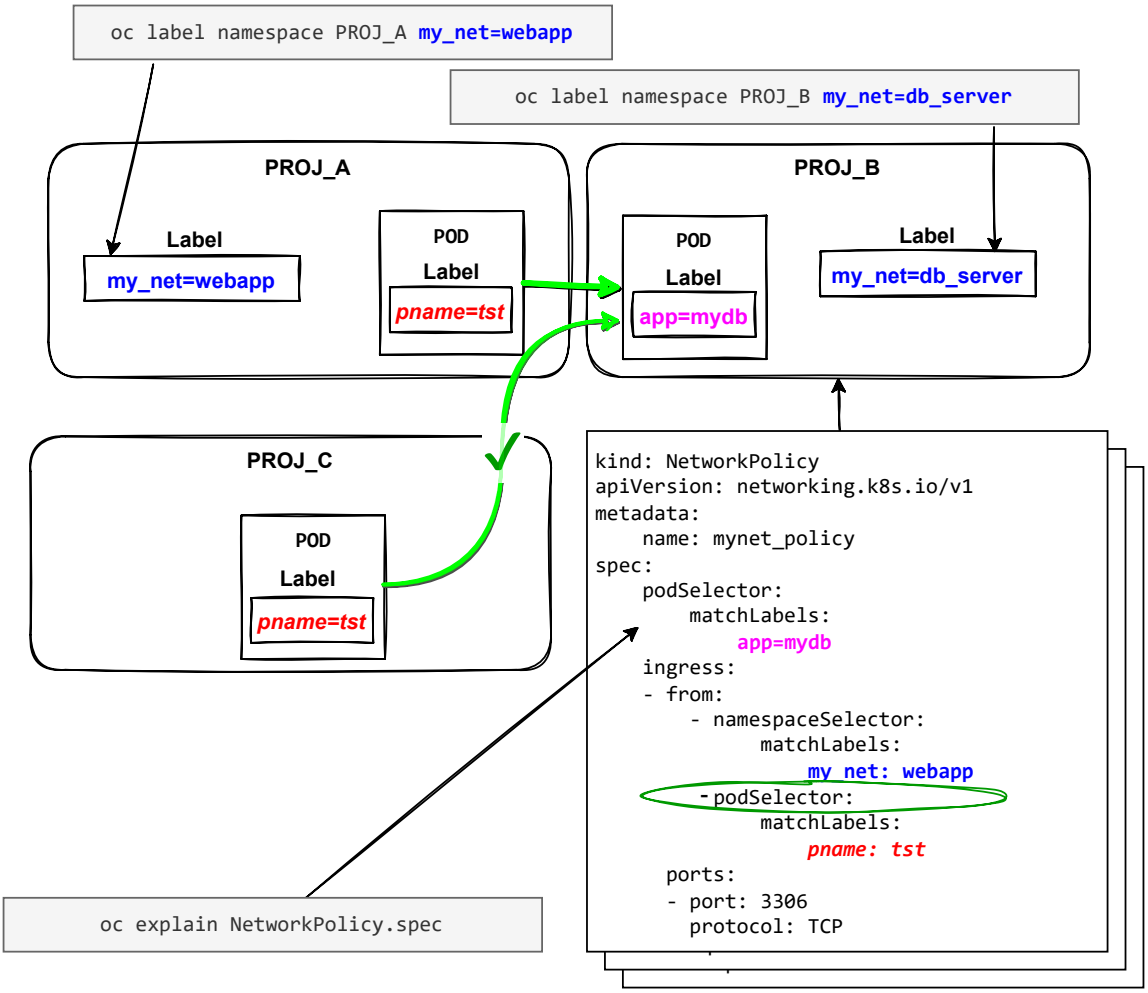
secret

Project: *openshift-config*

```
$ htpasswd -cBb ./myusers <USER> <PASSWORD>
$ oc create secret generic htpasswd-secret --from-file htpasswd=./myhtpasswd -n openshift-config
```

Termination Types





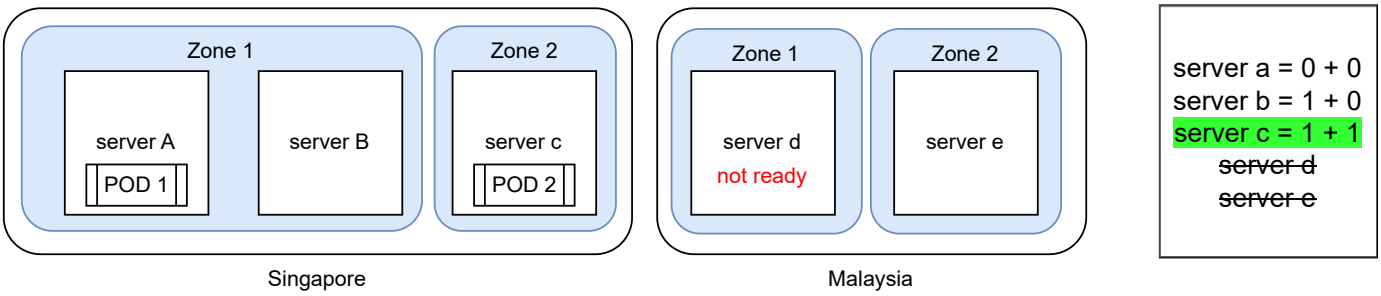
POD Scheduling

1. Get a list of all NODES

2. Go through all the predicates for FILTERing. If NODE fails predicate rule, remove from list. Region affinity.

3. With remainder list of NODES, prioritize them using the weightage rules. NO filtering of NODES done here. Zone anti-affinity.

4. Select the NODE with highest points.



`kubectl label node <NODE> <KEY>=<VALUE>`

Region

<KEY> = failure-domain.beta.kubernetes.io/region

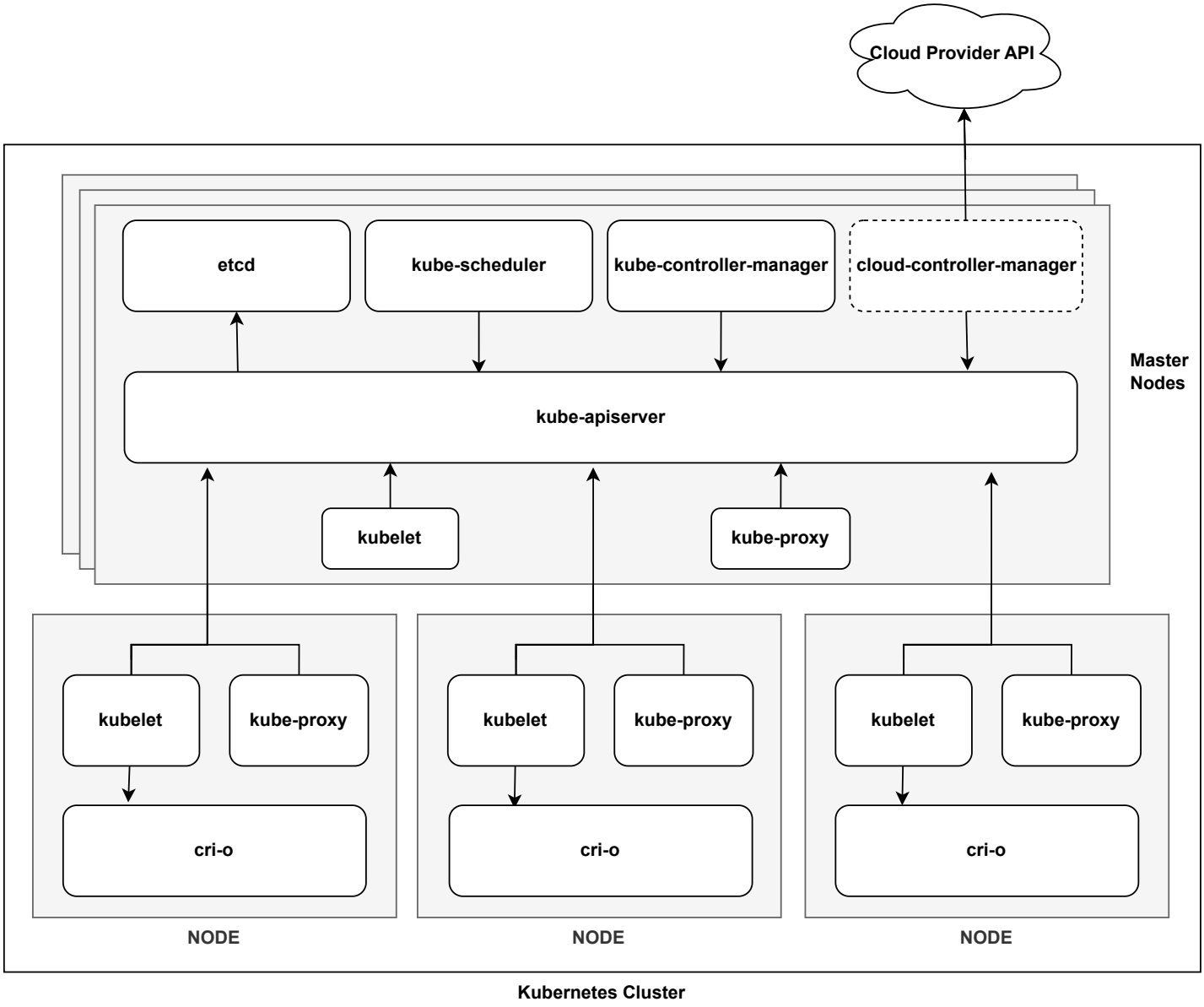
A set of hosts in closed geographical area. High speed connectivity.

Zone (availability zone)

<KEY> = failure-domain.beta.kubernetes.io/zone

A set of hosts that share common critical infra components (ups, switch, storage)

Upgrade Path Graph: https://access.redhat.com/labs/ocpupgradegraph/update_channel


Cloud Provider API
etcd
kube-scheduler
kube-controller-manager
cloud-controller-manager
kube-apiserver
kubelet
kube-proxy
Master Nodes
kubelet
kube-proxy
cri-o
kubelet
kube-proxy
cri-o
kubelet
kube-proxy
cri-o
NODE
NODE
NODE
Kubernetes Cluster