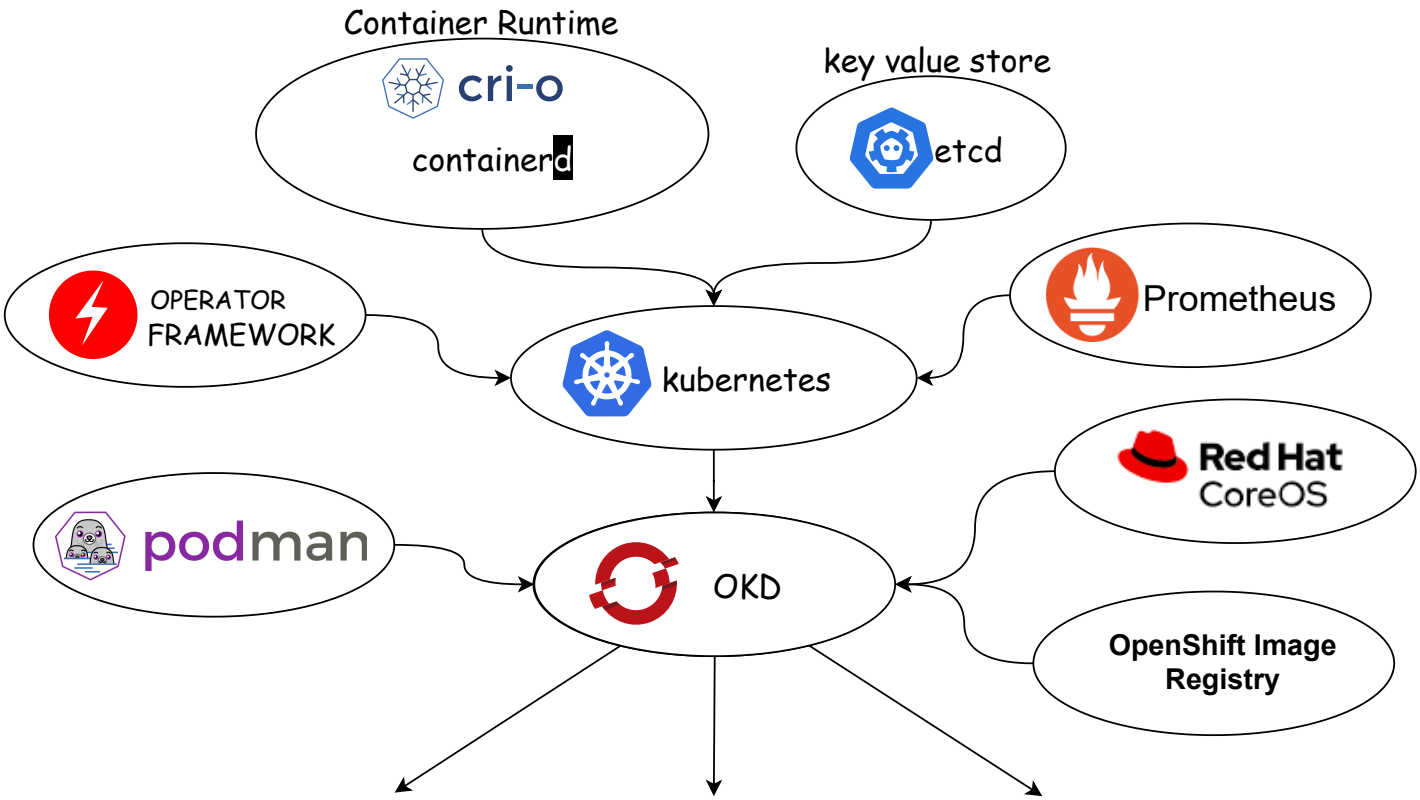


Red Hat OpenShift Container Platform

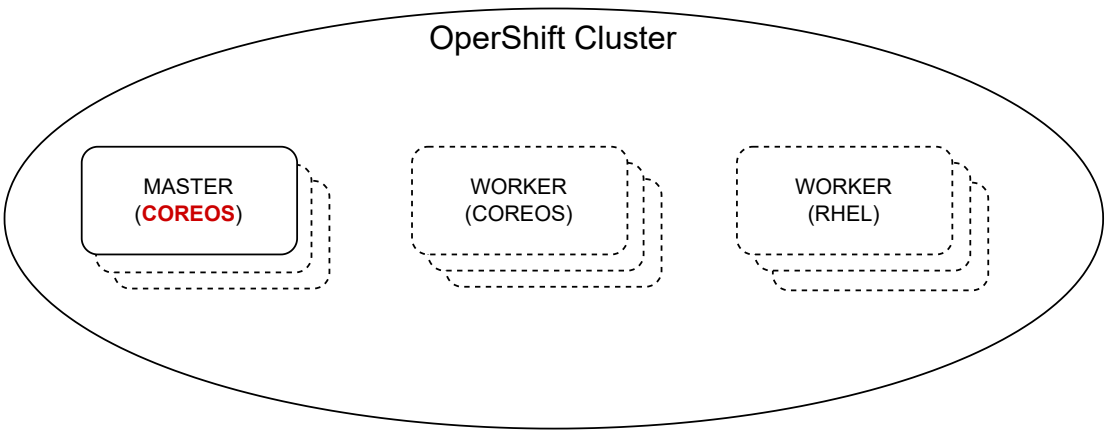
- Public/private DC.
- Bare metal and multiple cloud and virtualization providers.
- Full control by customer.

Red Hat OpenShift Dedicated

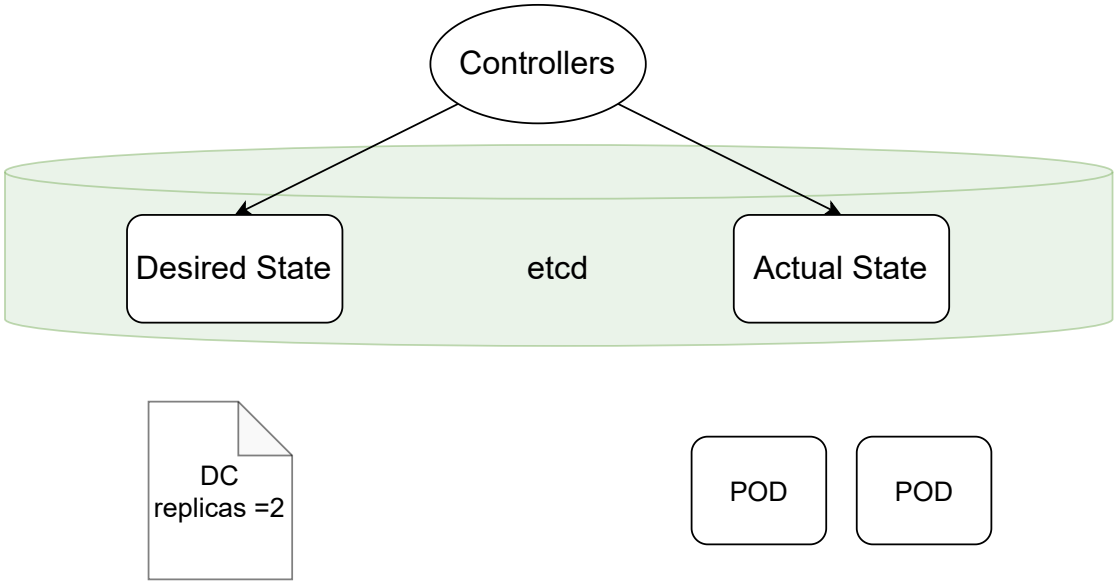
- Managed cluster in public cloud.
- RH manages the cluster.
- Customer manages updates and add-on services.

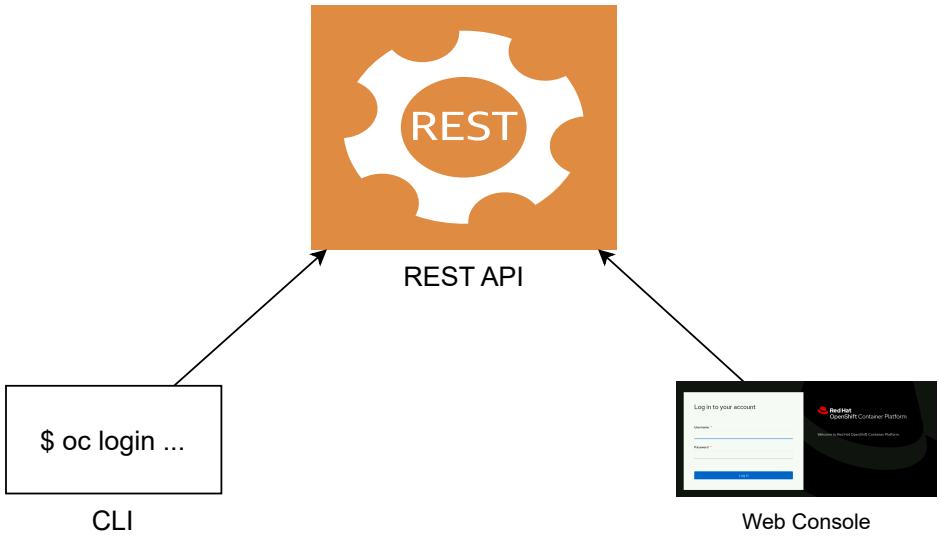
Red Hat OpenShift Online

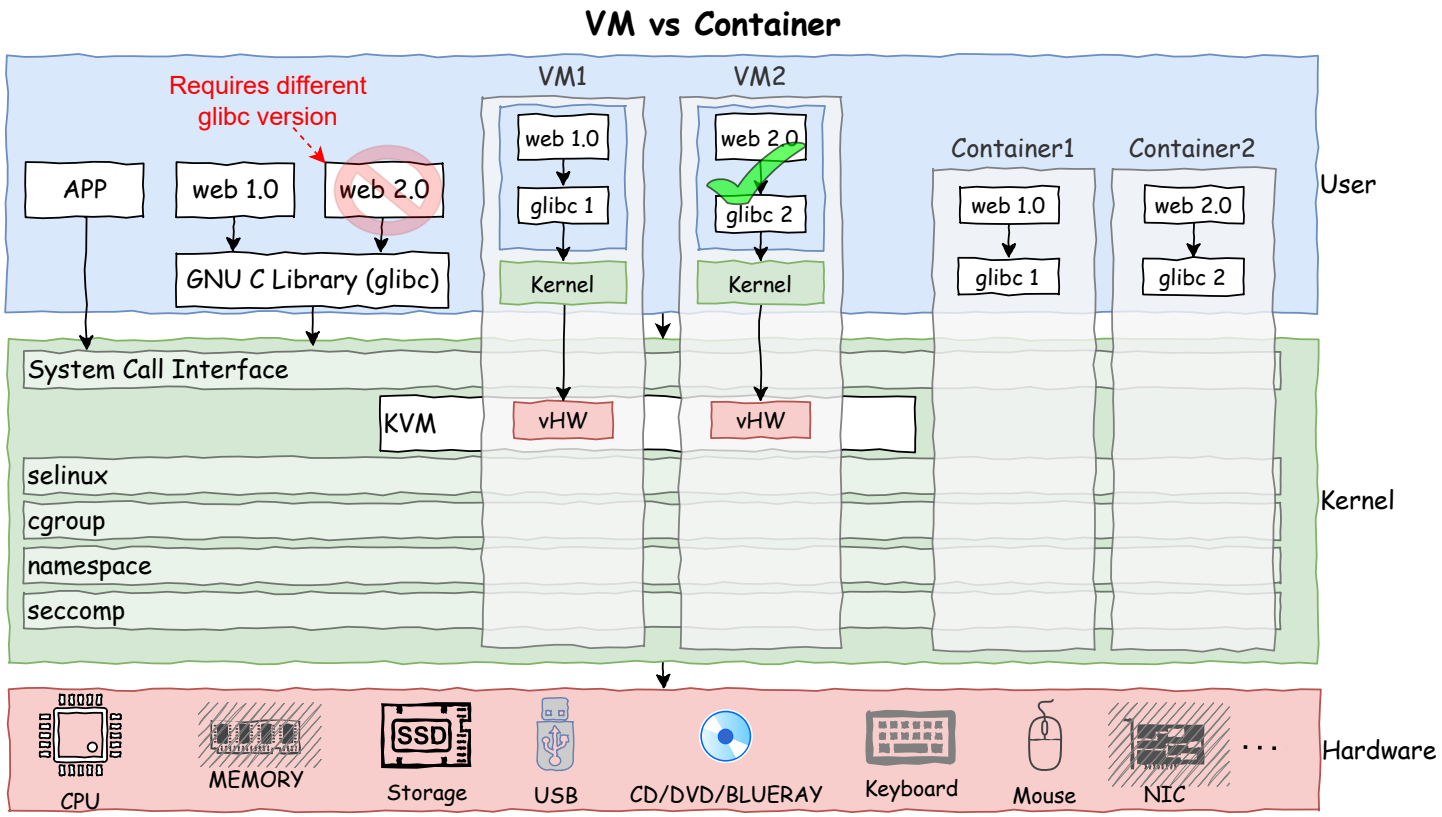
- Public hosted cluster.
- Shared resources by multiple customers.
- RH manages cluster life cycle.

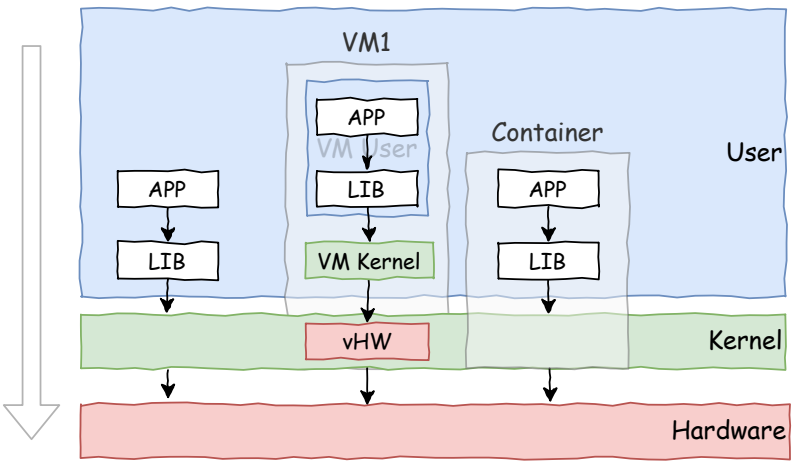


Kubernetes Declarative Architecture

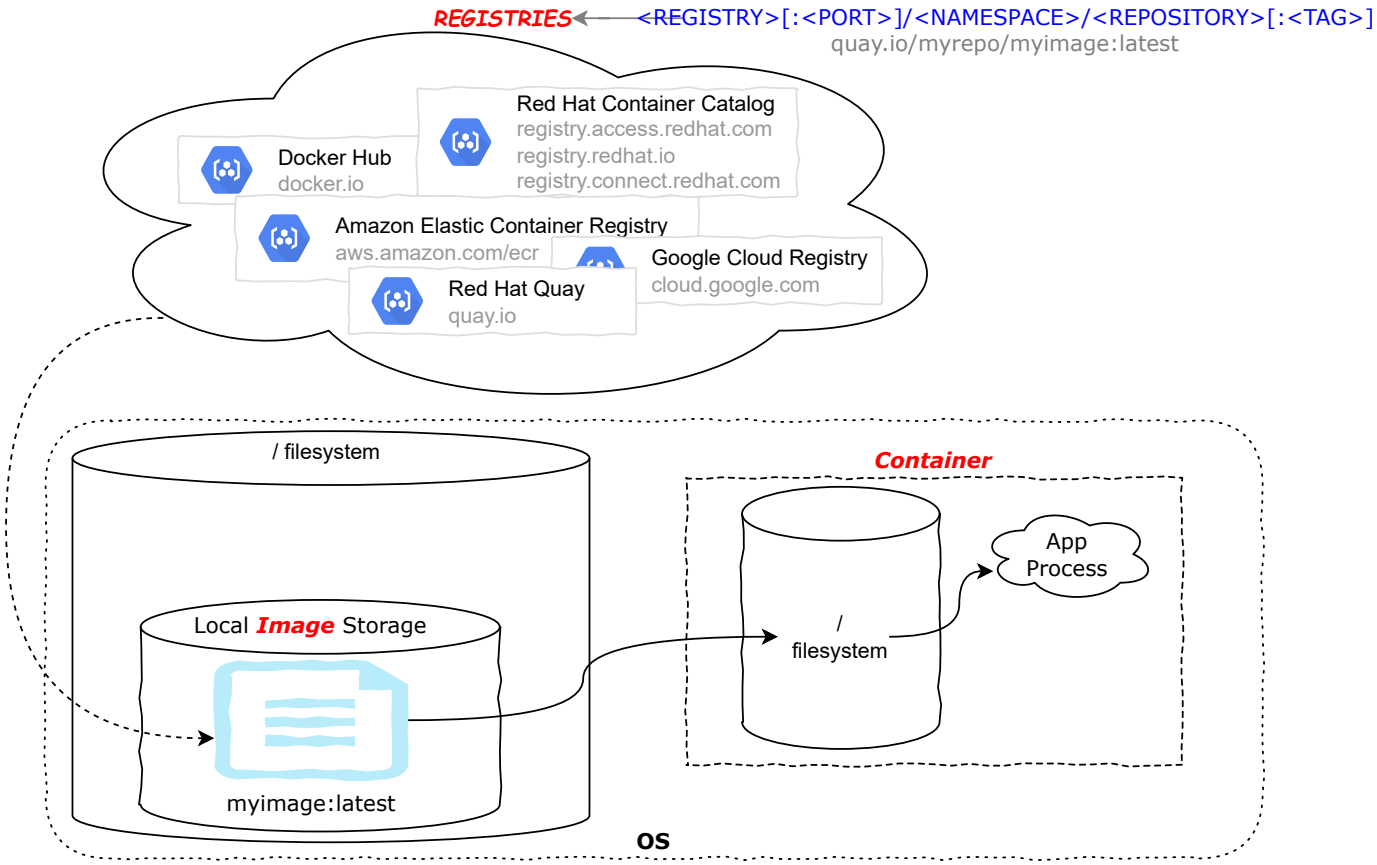




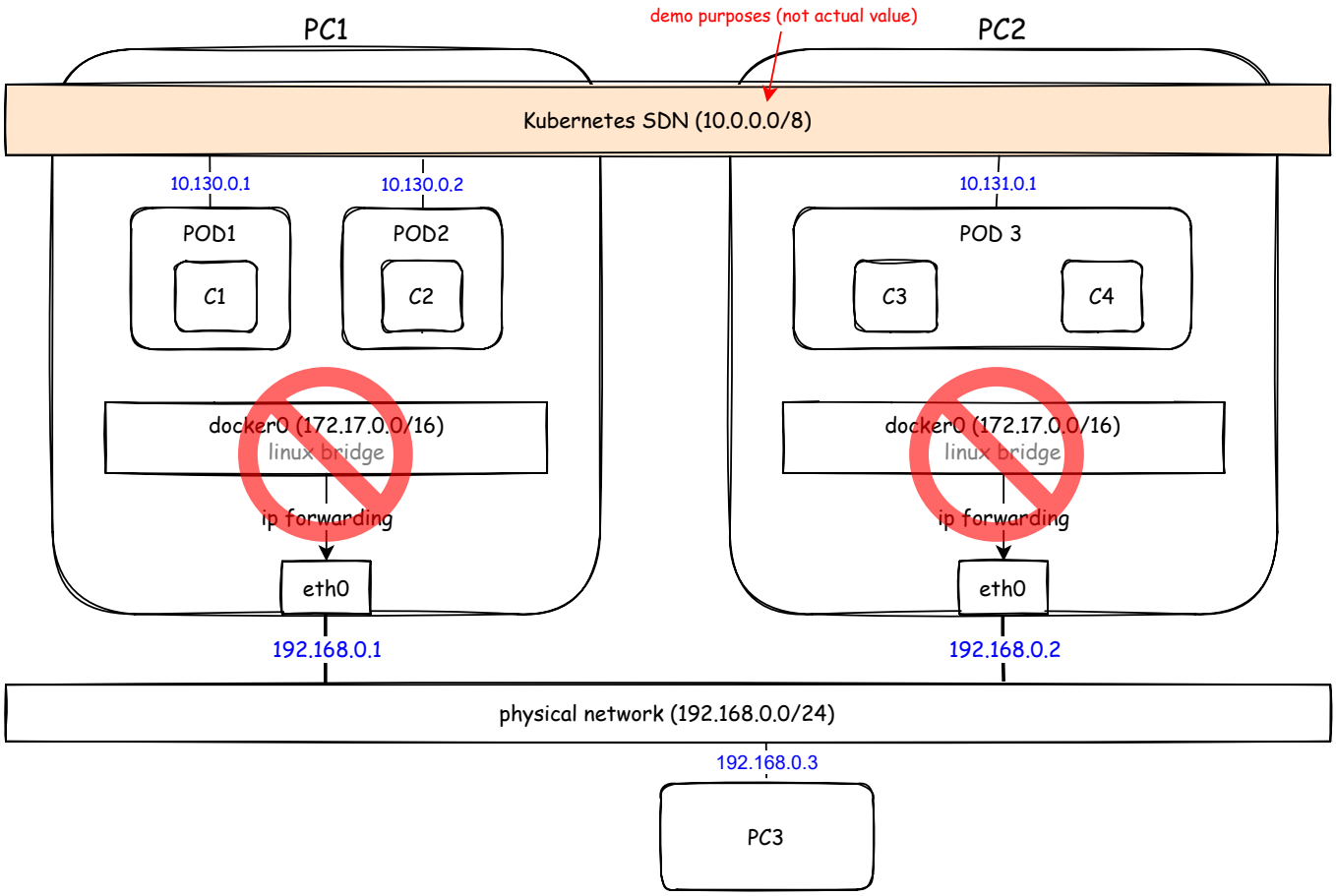

VM vs Container
Requires different glibc version
APP
web 1.0
web 2.0
GNU C Library (glibc)
VM1
web 1.0
glibc 1
Kernel
VM2
web 2.0
glibc 2
Kernel
Container1
web 1.0
glibc 1
Container2
web 2.0
glibc 2
User
System Call Interface
KVM
vHW
vHW
selinux
cgroup
namespace
seccomp
Kernel
CPU
MEMORY
SSD
Storage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

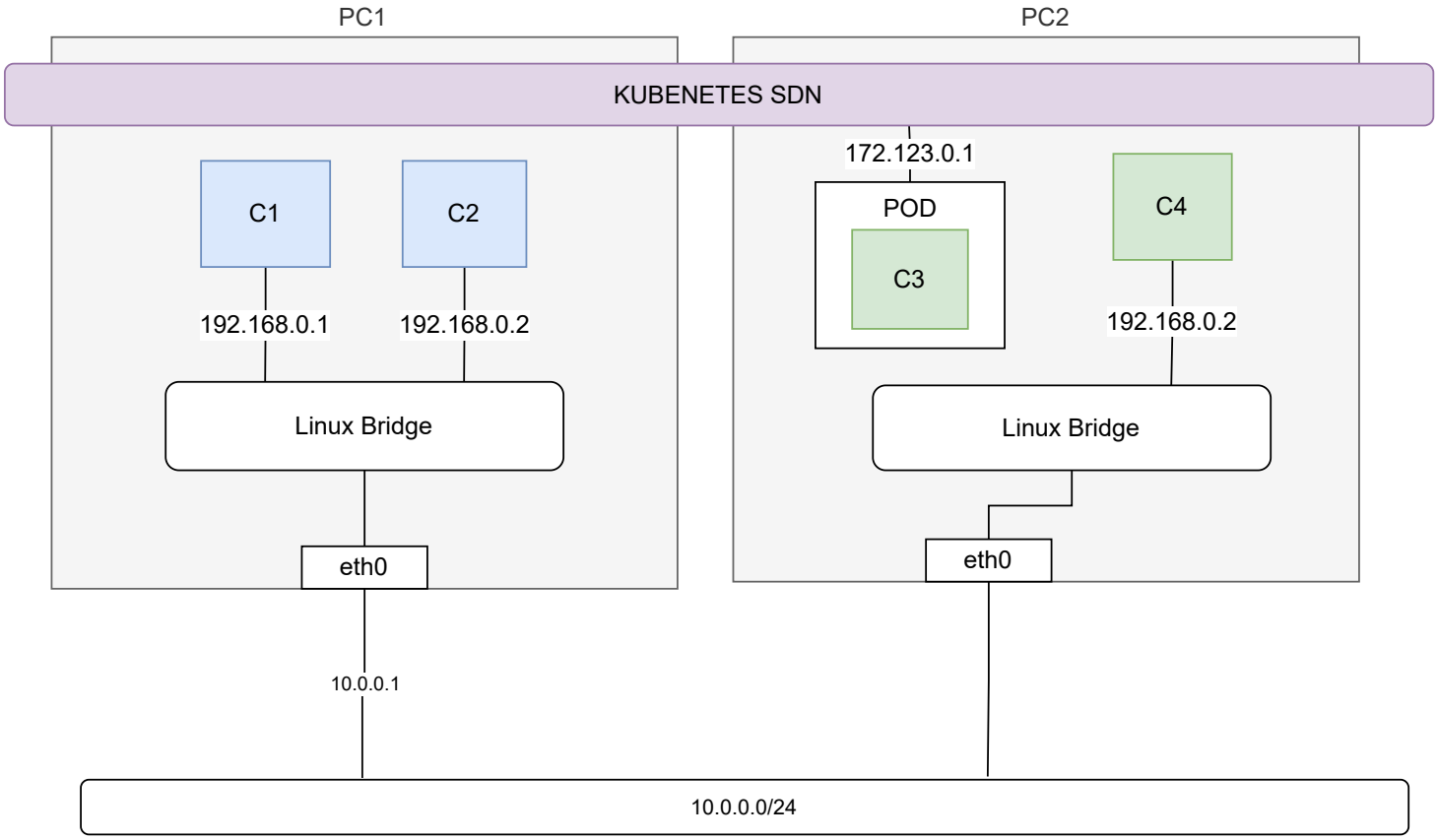

VM1
APP
VM User
LIB
VM Kernel
Container
APP
LIB
APP
LIB
User
vHW
Kernel
Hardware

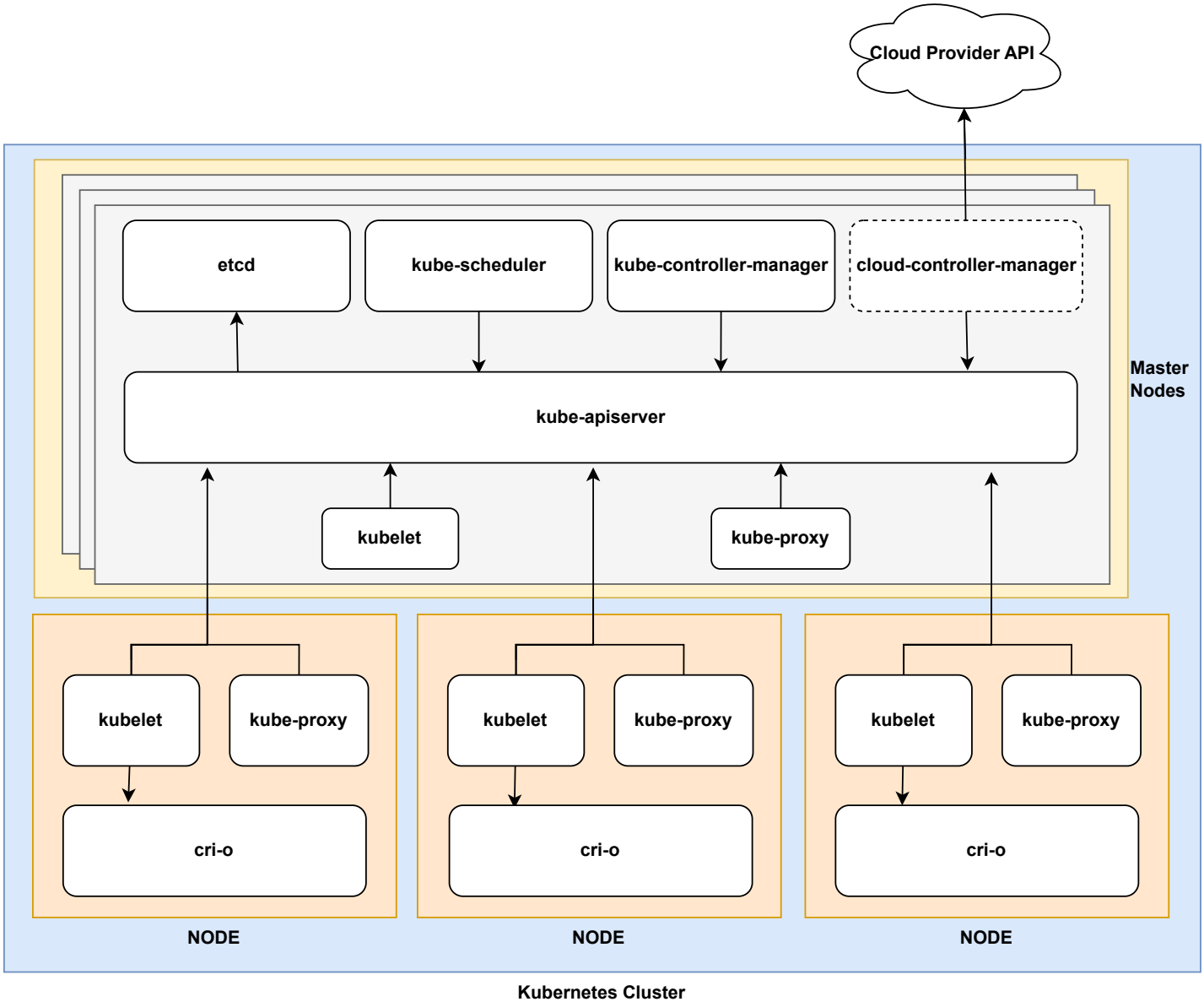
Container Architecture



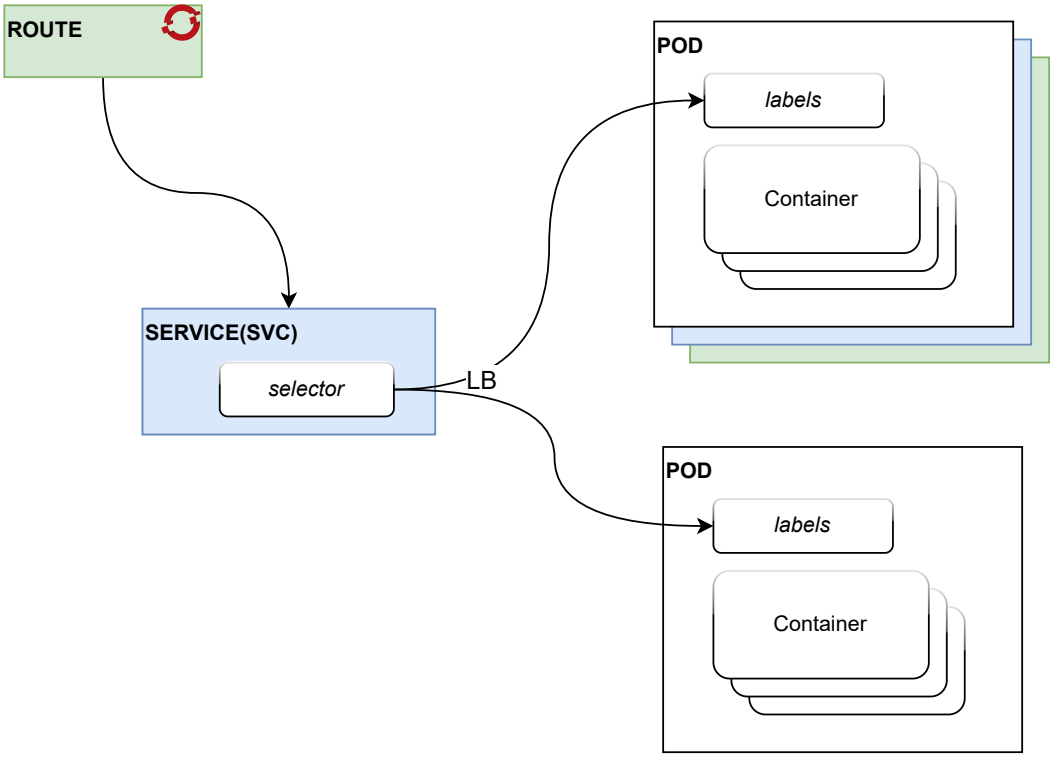
Basic Network - Container vs Kubernetes






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

Route, Service and Pod Relationship



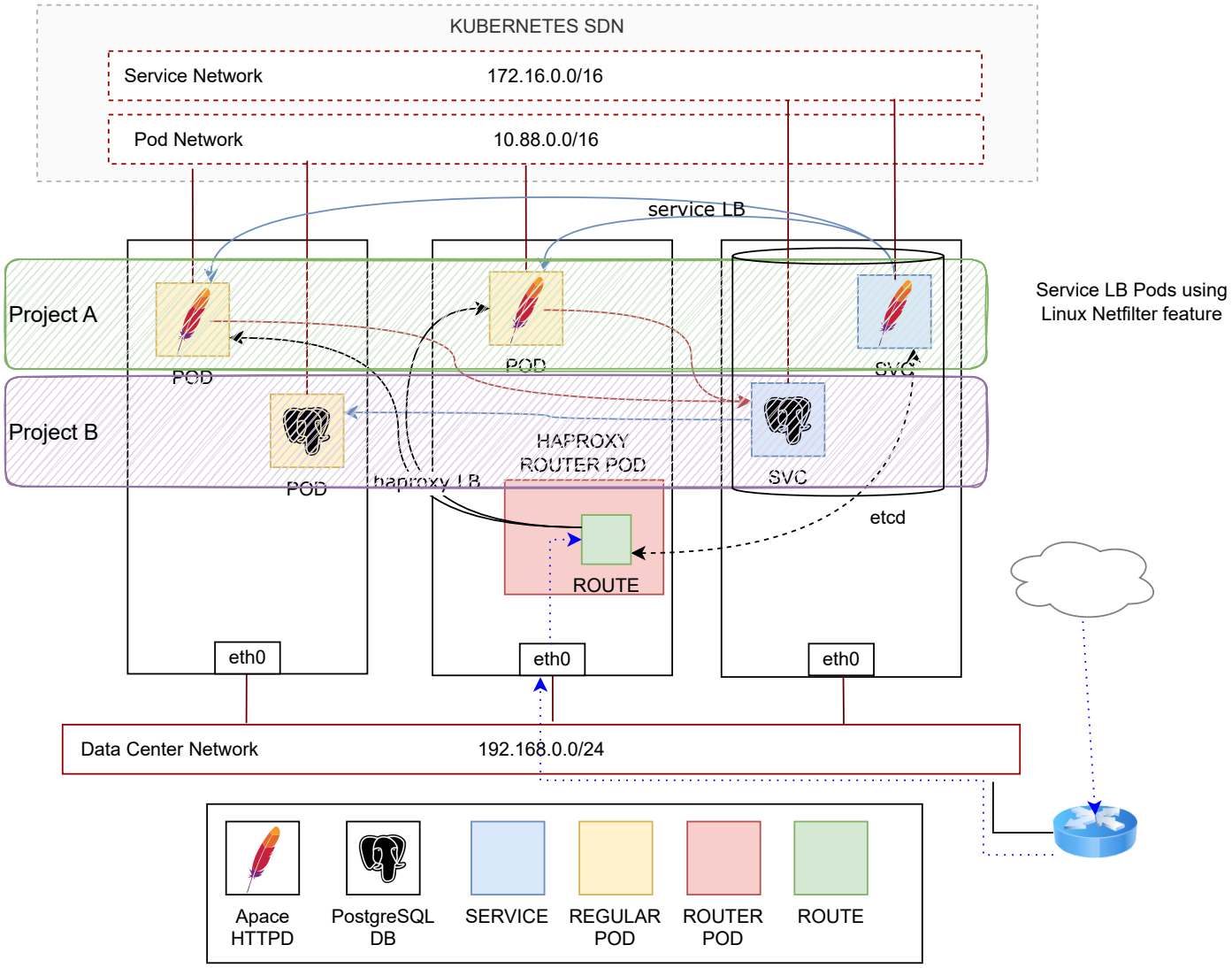
POD
A pod contains one or more containers.

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

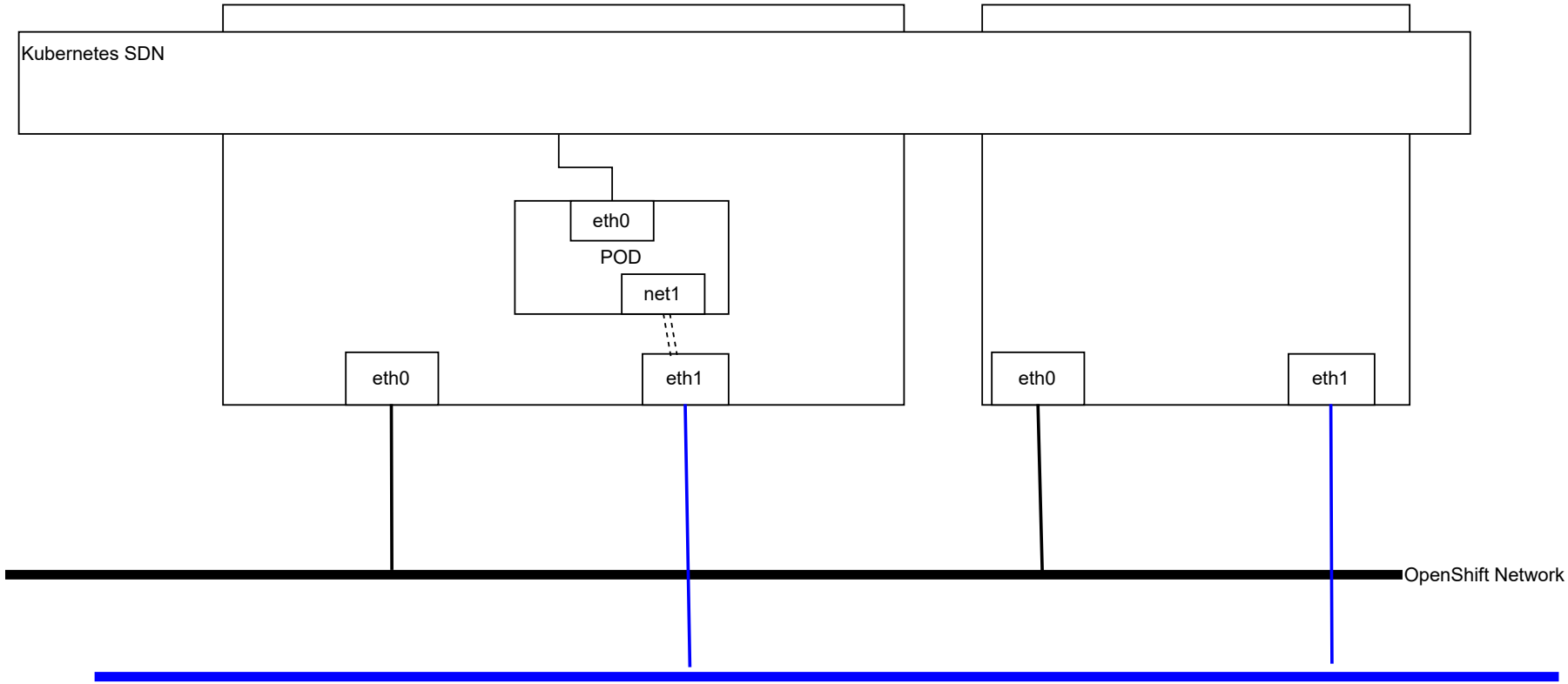
ROUTE
A route exposes the service to the external world.

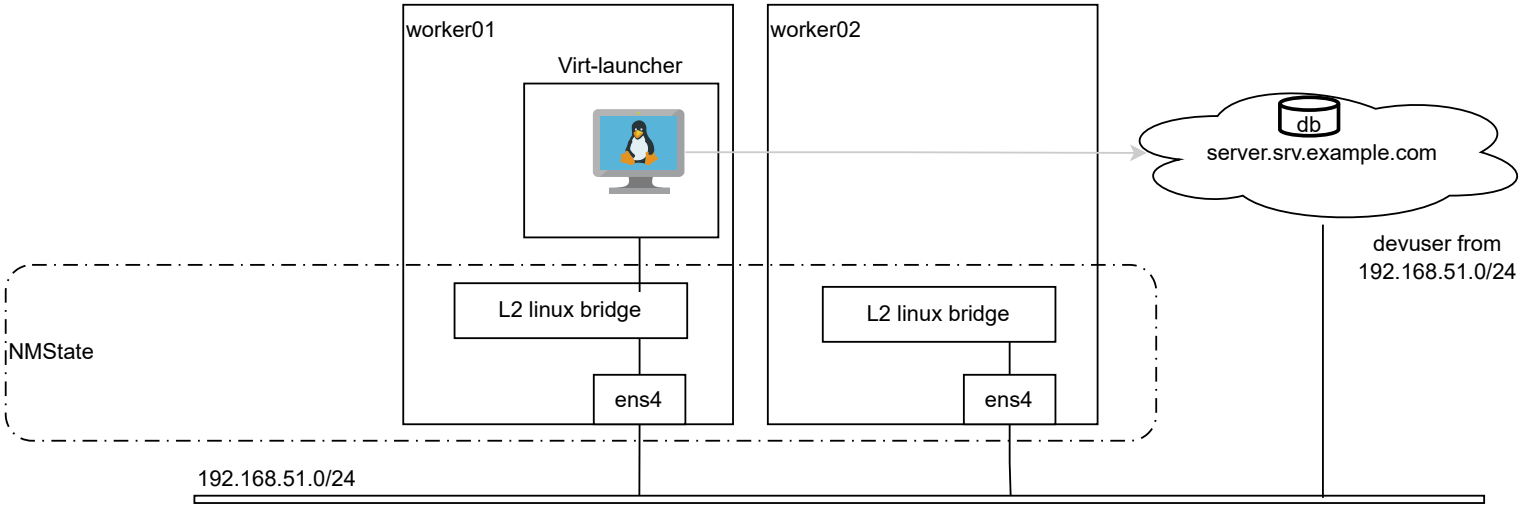
Warning: A service "can" refer to different pods, if the pods have the same label.

Sample of how Services are used

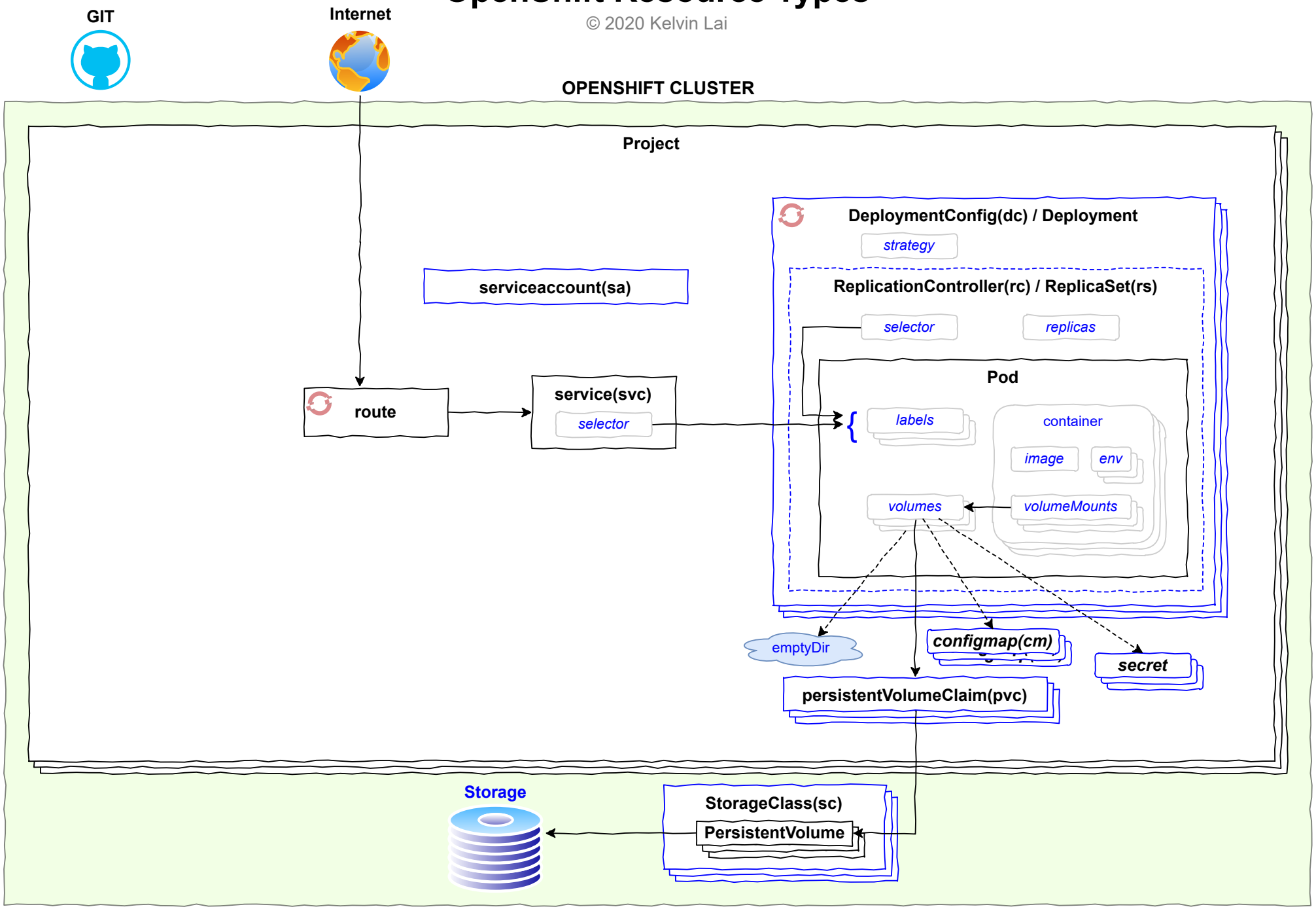


Service LB Pods using Linux Netfilter feature


Kubernetes SDN
eth0
POD
net1
eth0
eth1
eth0
eth1
OpenShift Network


worker01
Virt-launcher
worker02
db
server.srv.example.com
devuser from
192.168.51.0/24
L2 linux bridge
L2 linux bridge
NMState
ens4
ens4
192.168.51.0/24

OpenShift Resource Types



Deploying Applications with OpenShift

Methods to create applications:

1. Using existing containerised applications

```
oc new-app --docker-image=<IMAGE>
```

2. From Source Code using S2I

```
oc new-app <URL>
```

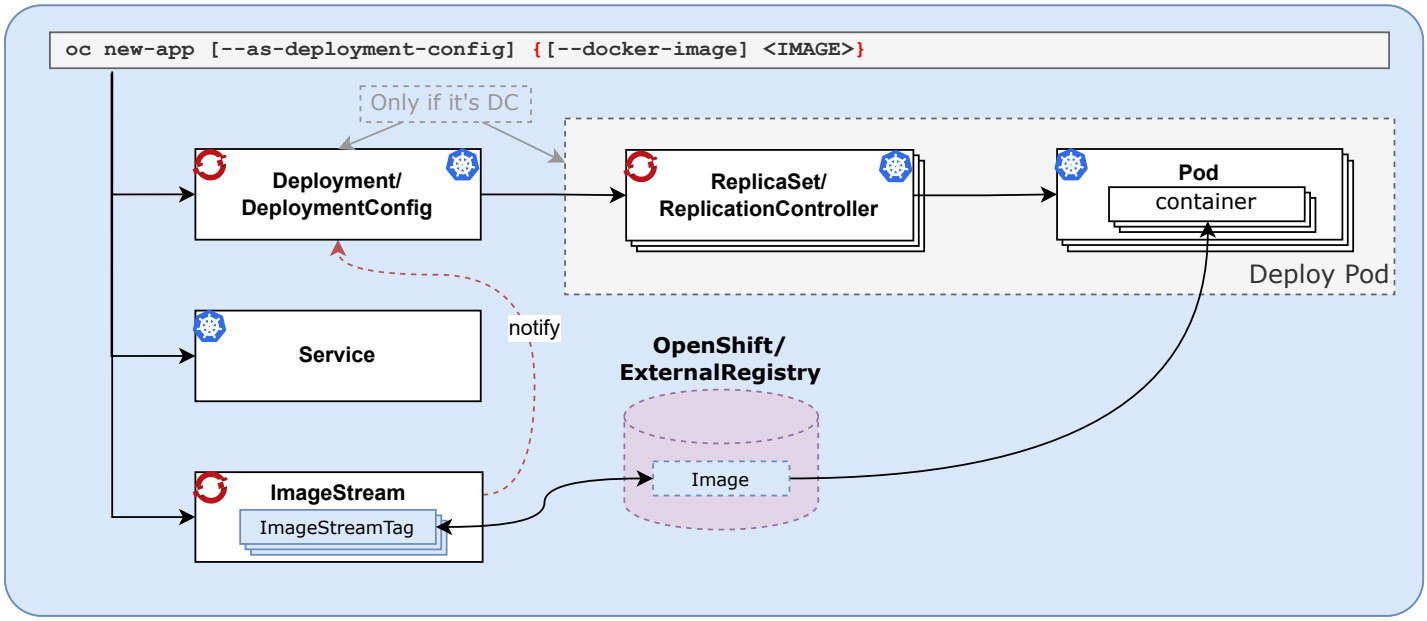
3. Using yaml/json file

```
oc new-app -f <FILE>.yaml
```

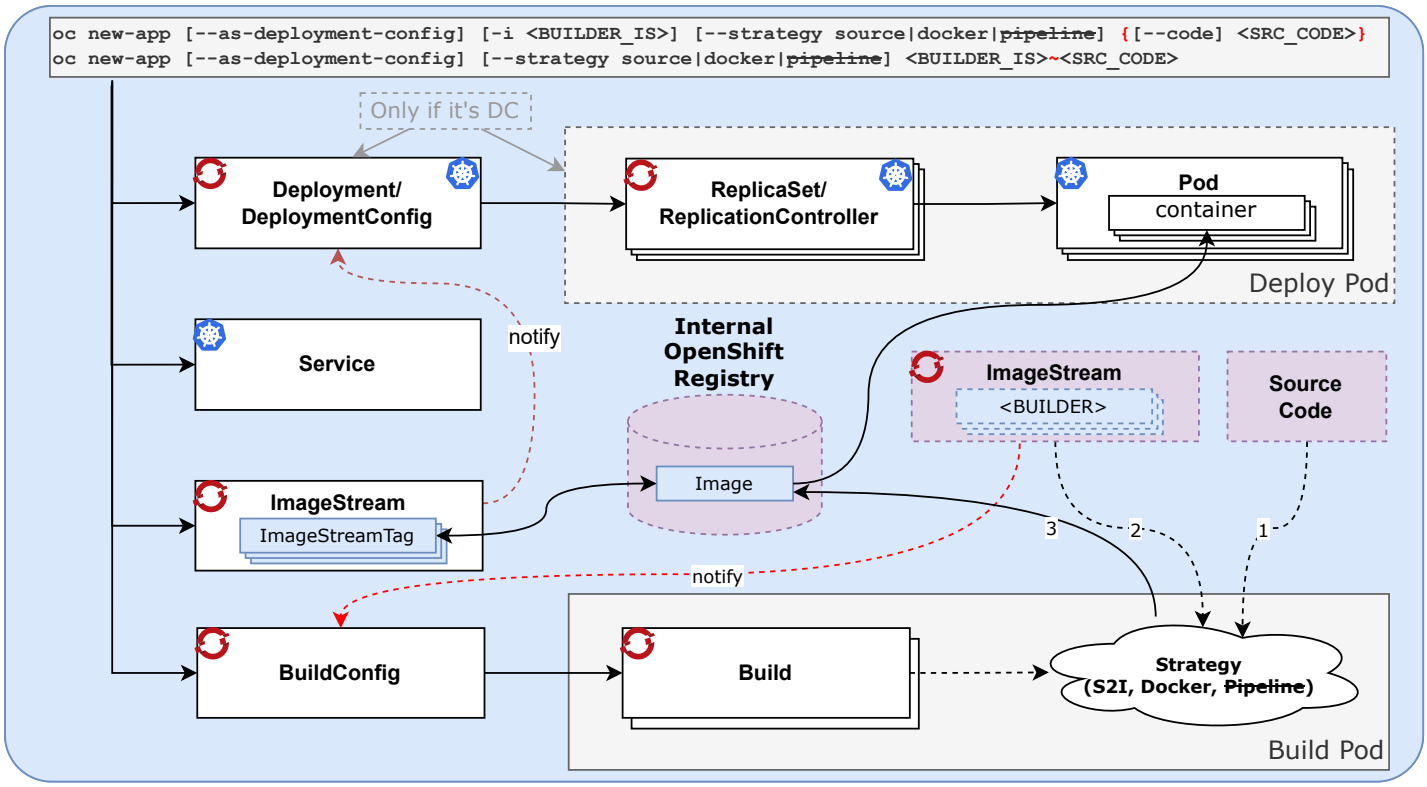
4. Using template

```
oc new-app --template=<TEMPLATE> --param=<PARAM> --param-file=<PARAM_FILE>
```

1. Use Existing Image



2. Managed Life Cycle



```
oc new-app -i myphp https://github.com/user/myapp#branch --context-dir <DIR>
```

```
oc new-app -i myphp:7.1 https://github.com/user/myapp
```

```
oc new-app myphp:7.1~https://github.com/user/myapp
```

NOTE: -i option needs git client to be installed

Options

-o json|yaml	inspect resource definitions without creating
--name <NAME>	adds a label "app=<NAME>" to all resources, Use `oc delete all -l "app=<NAME>"` to cleanup

IMPORT IMAGES

```
oc import-image <IMG_STREAM> [--confirm] --from <IMAGE> [--insecure]
where,
    <IMAGE> = <REGISTRY>[:<PORT>]/<NAMESPACE>/<REPOSITORY>[:<TAG>]
```

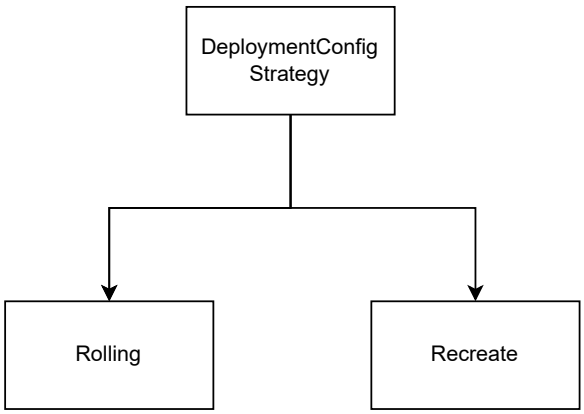
oc new-app command in OpenShift 4.5 makes use of *deployment* resource. Use --as-deployment-config if you wish to create *deployment config* instead.

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

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

oc expose svc <SVC_NAME> [--name <ROUTE_NAME>] [--hostname <FQDN>]

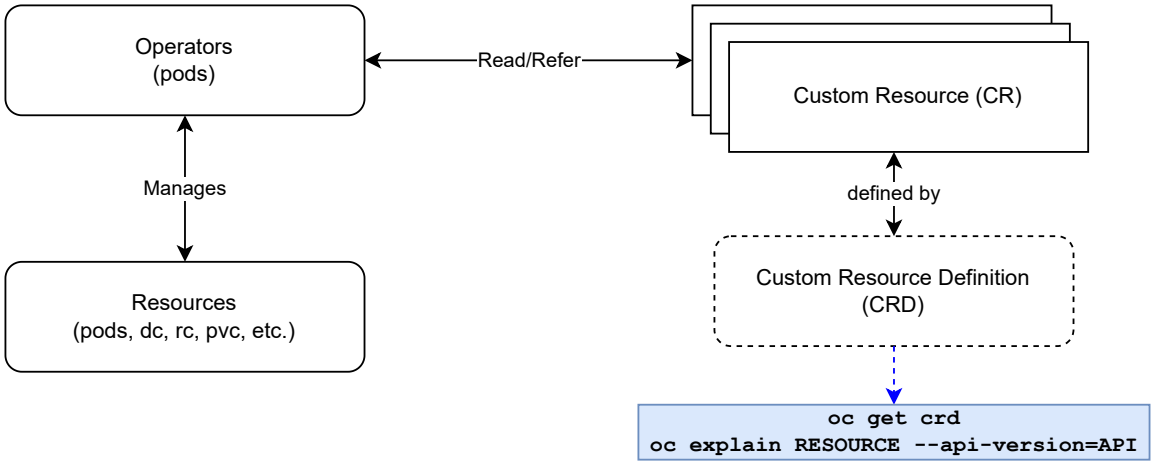
DNS DEFAULT NAME = <ROUTE_NAME>-<PROJ>.<DOMAIN WILDCARD>
<DOMAIN_WILDCARD> = apps.<BASE_DOMAIN>

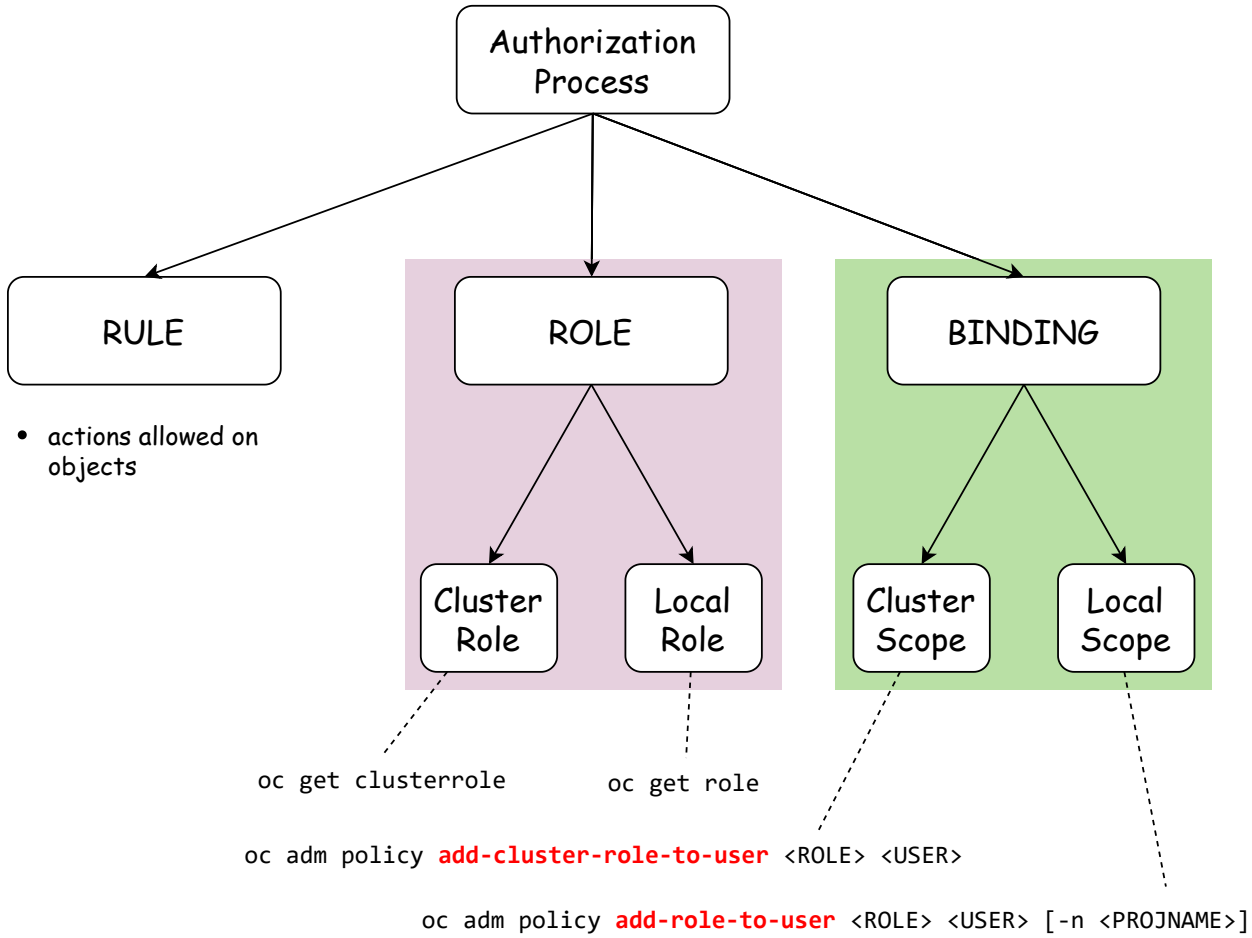



```
spec:
  strategy:
    type: Rolling
    rollingParams:
      pre: {}
      post: {}
```

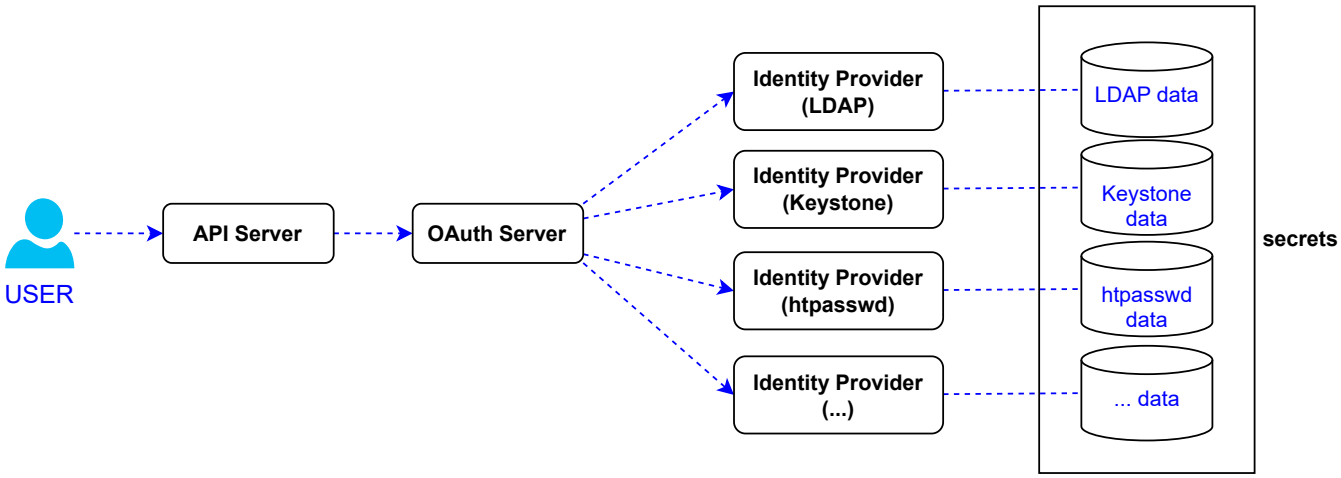
```
spec:
  strategy:
    type: Recreate
    recreateParams:
      pre: {}
      mid: {}
      post: {}
```

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*

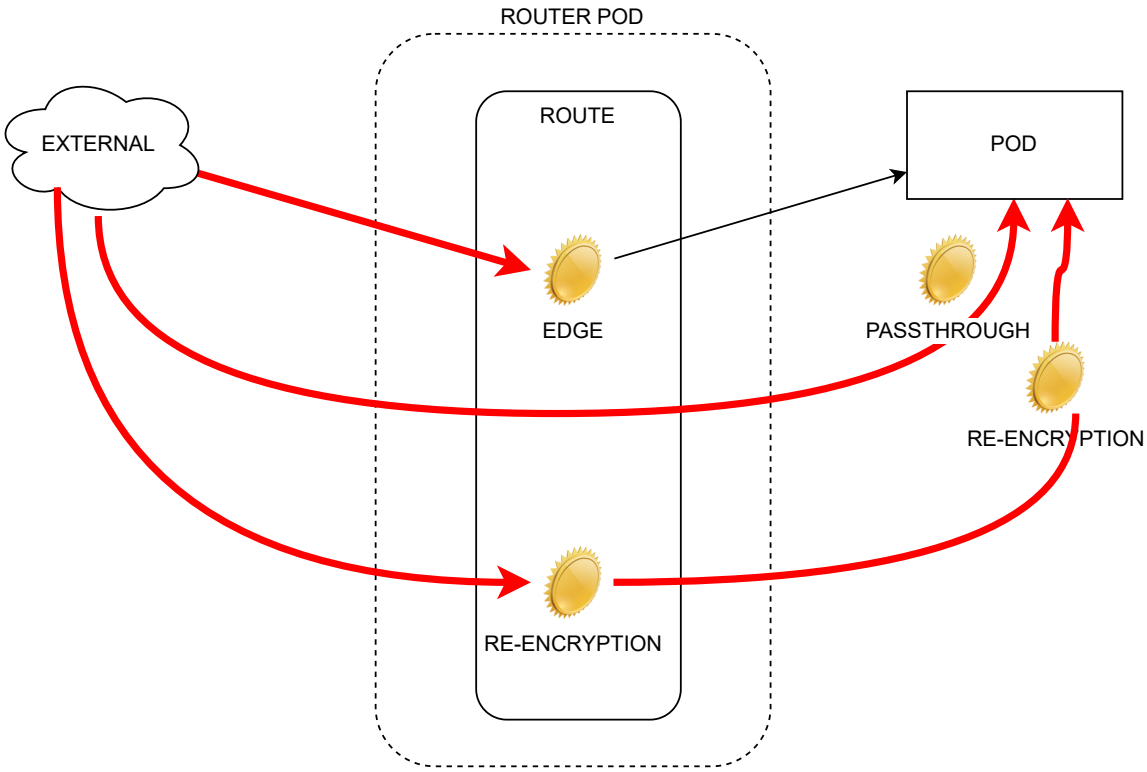
```
$ oc get pods -n openshift-authentication
```

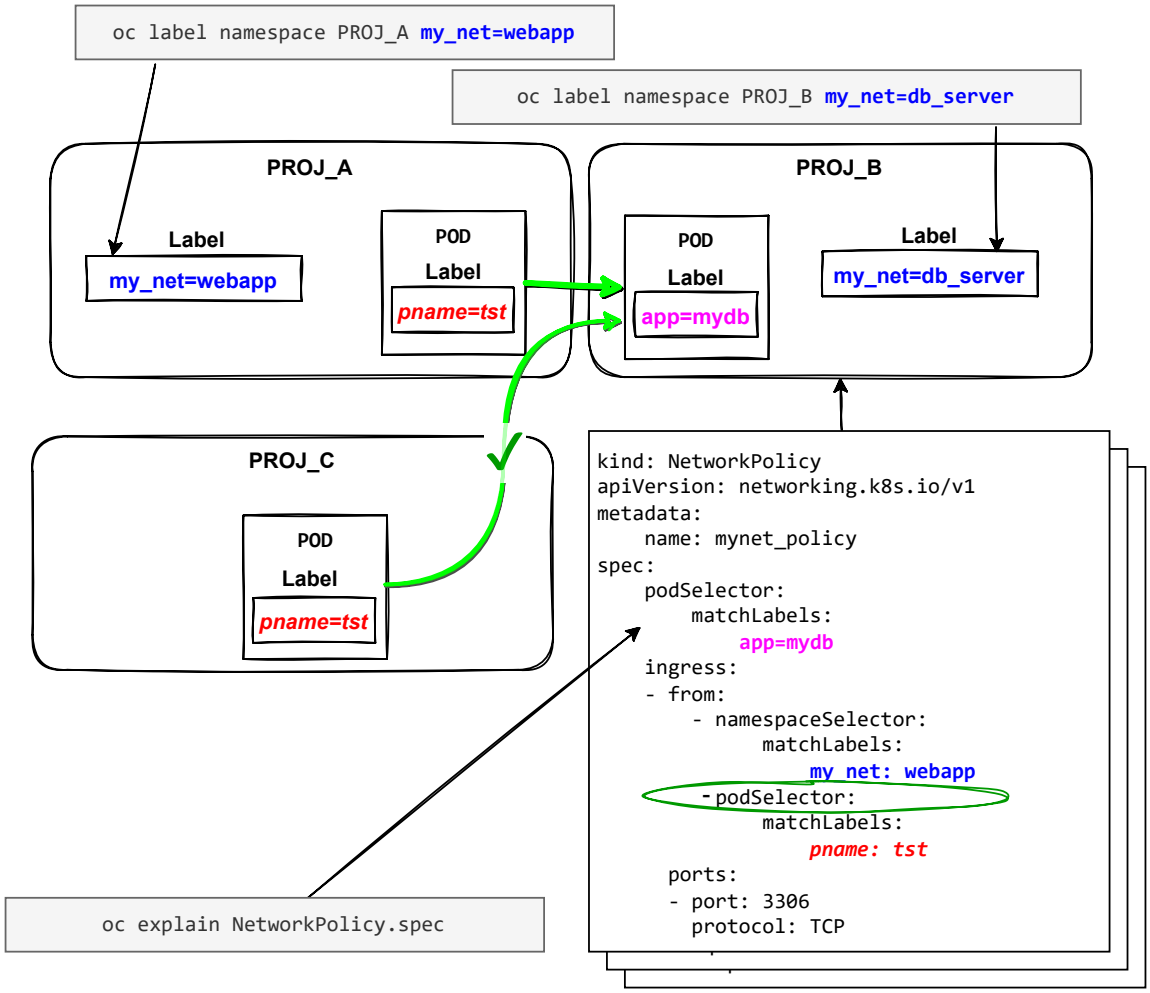
```
$ oc get oauth cluster -o yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
  ...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: htpasswd-secret
    mappingMethod: claim
    name: htpasswd_provider
    type: HTPasswd
```

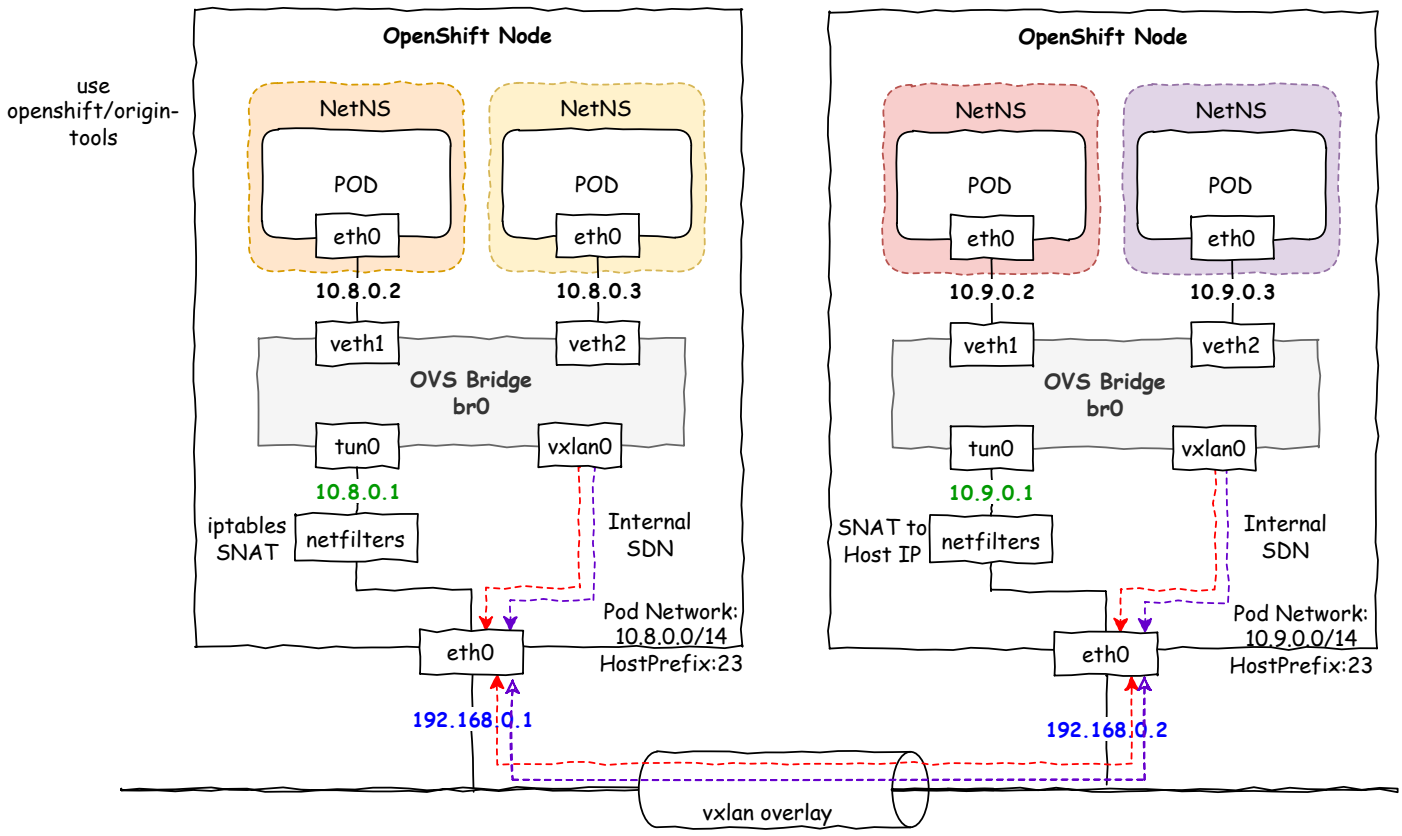
secret

Project: *openshift-config*

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


ROUTER POD
EXTERNAL
ROUTE
POD
EDGE
PASSTHROUGH
RE-ENCRYPTION
RE-ENCRYPTION


oc label namespace PROJ_A my_net=webapp
oc label namespace PROJ_B my_net=db_server
PROJ_A
Label
my_net=webapp
POD
Label
pname=tst
PROJ_B
POD
Label
app=mydb
Label
my_net=db_server
PROJ_C
POD
Label
pname=tst
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
name: mynet_policy
spec:
podSelector:
matchLabels:
app=mydb
ingress:
- from:
- namespaceSelector:
matchLabels:
my_net: webapp
- podSelector:
matchLabels:
pname: tst
ports:
- port: 3306
protocol: TCP
oc explain NetworkPolicy.spec



nnnnnnnn . nnnnnnxx . xxxxxxxx . xxxxxxxx

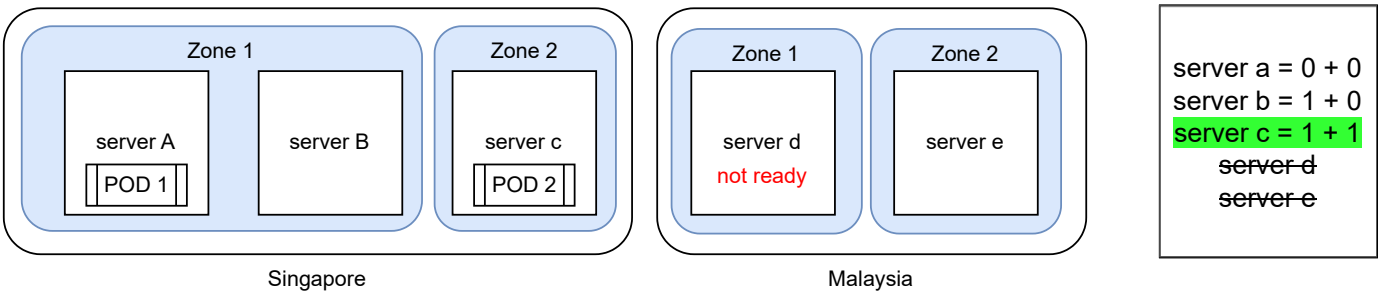
POD Scheduling

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.



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

Region

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

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

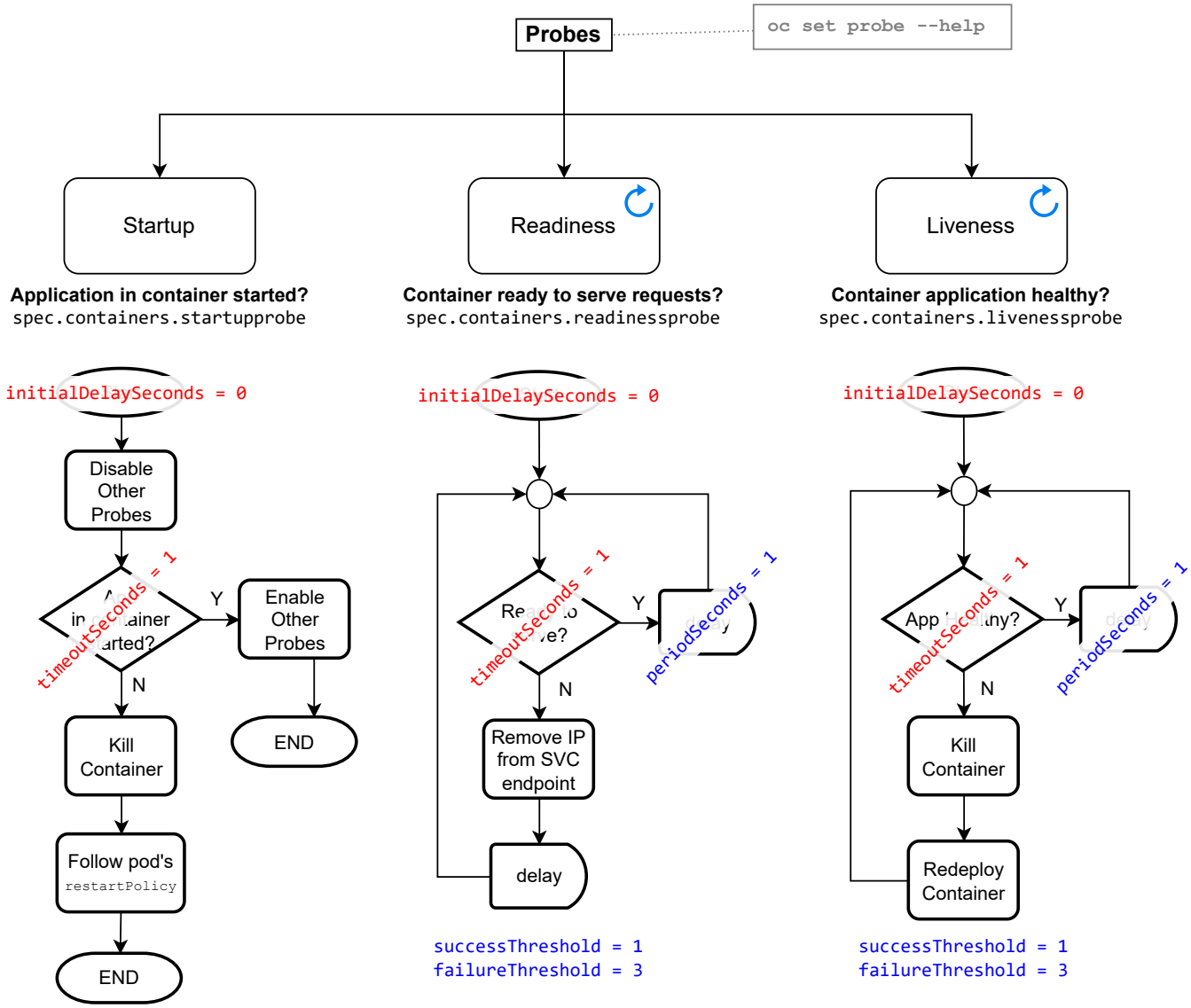
Zone (availability zone)

<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

Health Monitoring



successThreshold = 1
failureThreshold = 3

Mandatory

Optional

Deployment/DC

```
spec:
  template:
    spec:
      containers:
        - image: ...
          ...Probe:
```

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 15
  timeoutSeconds: 1
```

```
livenessProbe:
  exec:
    command:
    - cat
    - /tmp/health
  initialDelaySeconds: 15
  timeoutSeconds: 1
```

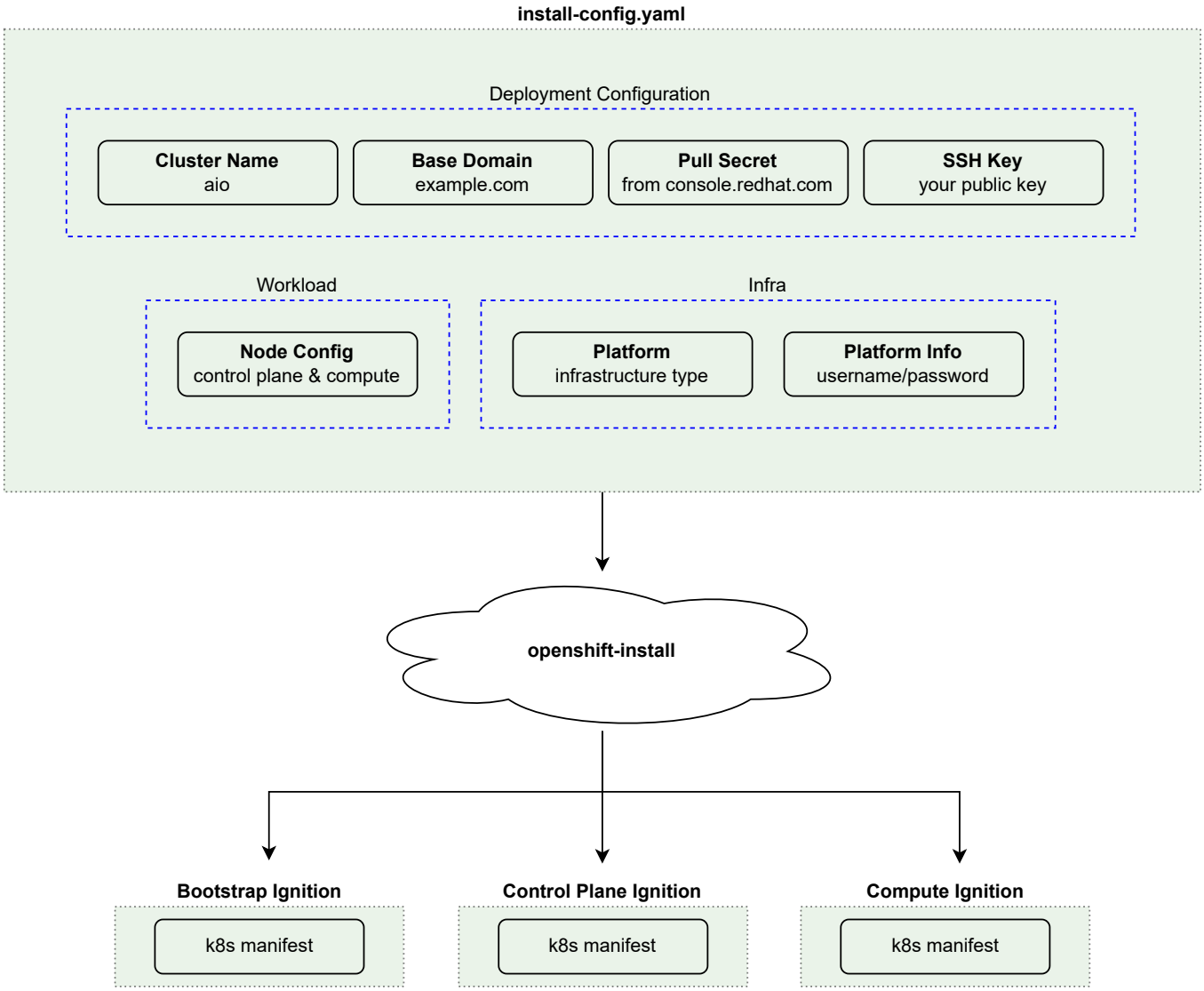
```
livenessProbe:
  tcpSocket:
    port: 3306
  periodSeconds: 15
  timeoutSeconds: 1
```

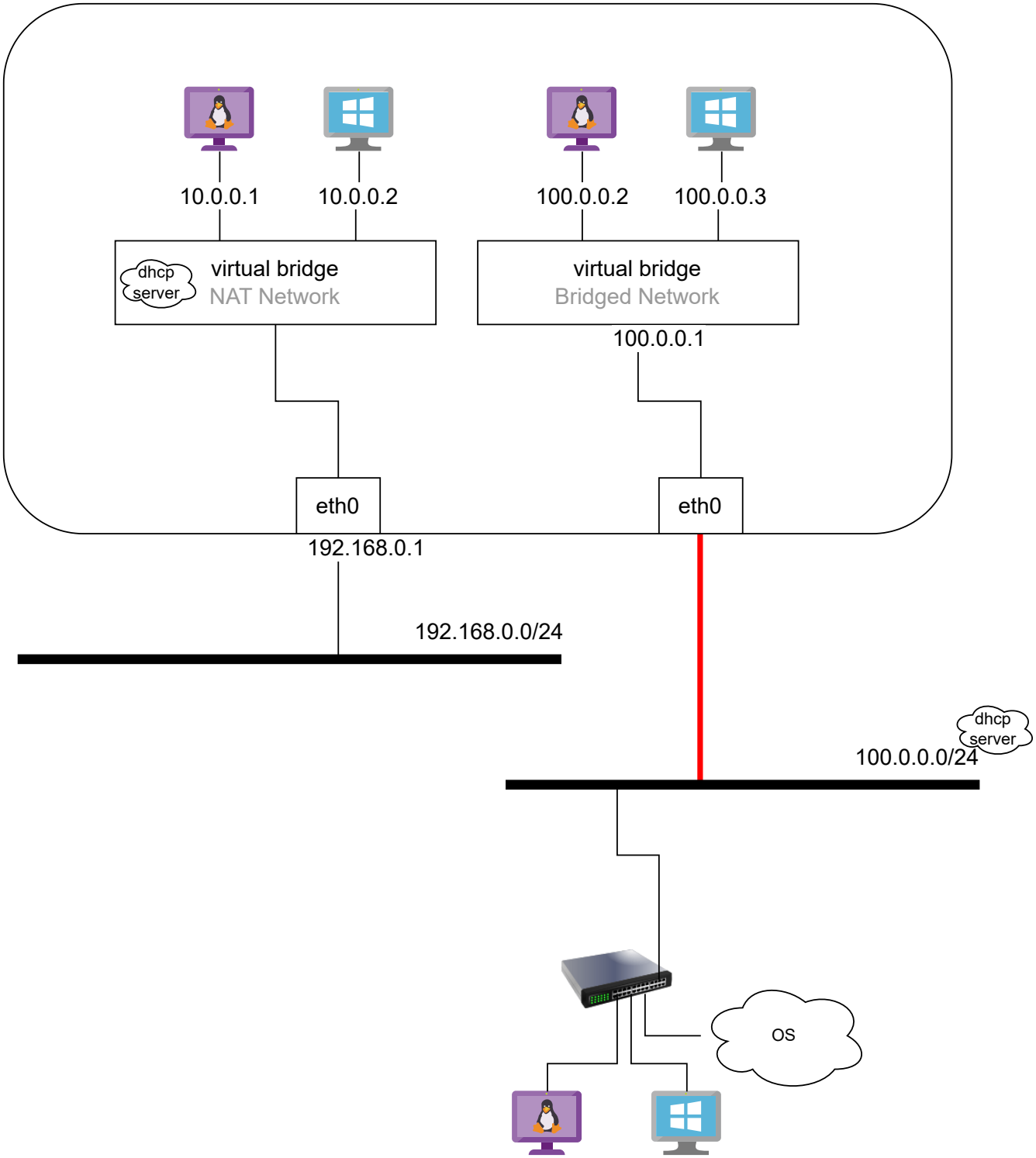
```
oc set probe deployment test-php \
    --readiness \
    --get-url=http://:8080/ready  \
    --initial-delay-seconds=15 \
    --timeout-seconds=1
```

```
oc set probe deployment test-php \
    --liveness \
    -- cat /tmp/health  \
    --initial-delay-seconds=15 \
    --timeout-seconds=1
```

```
oc set probe deployment test-php \
    --liveness \
    --open-tcp 3306
    --period-seconds 15 \
    --timeout-seconds 1
```

Extra - https://cloud.redhat.com/blog/liveness-and-readiness-probes


install-config.yaml
Deployment Configuration
Cluster Name
aio
Base Domain
example.com
Pull Secret
from console.redhat.com
SSH Key
your public key
Workload
Node Config
control plane & compute
Infra
Platform
infrastructure type
Platform Info
username/password
openshift-install
Bootstrap Ignition
k8s manifest
Control Plane Ignition
k8s manifest
Compute Ignition
k8s manifest


10.0.0.1
10.0.0.2
100.0.0.2
100.0.0.3
dhcp server
virtual bridge
NAT Network
virtual bridge
Bridged Network
100.0.0.1
eth0
eth0
192.168.0.1
192.168.0.0/24
dhcp server
100.0.0.0/24
OS