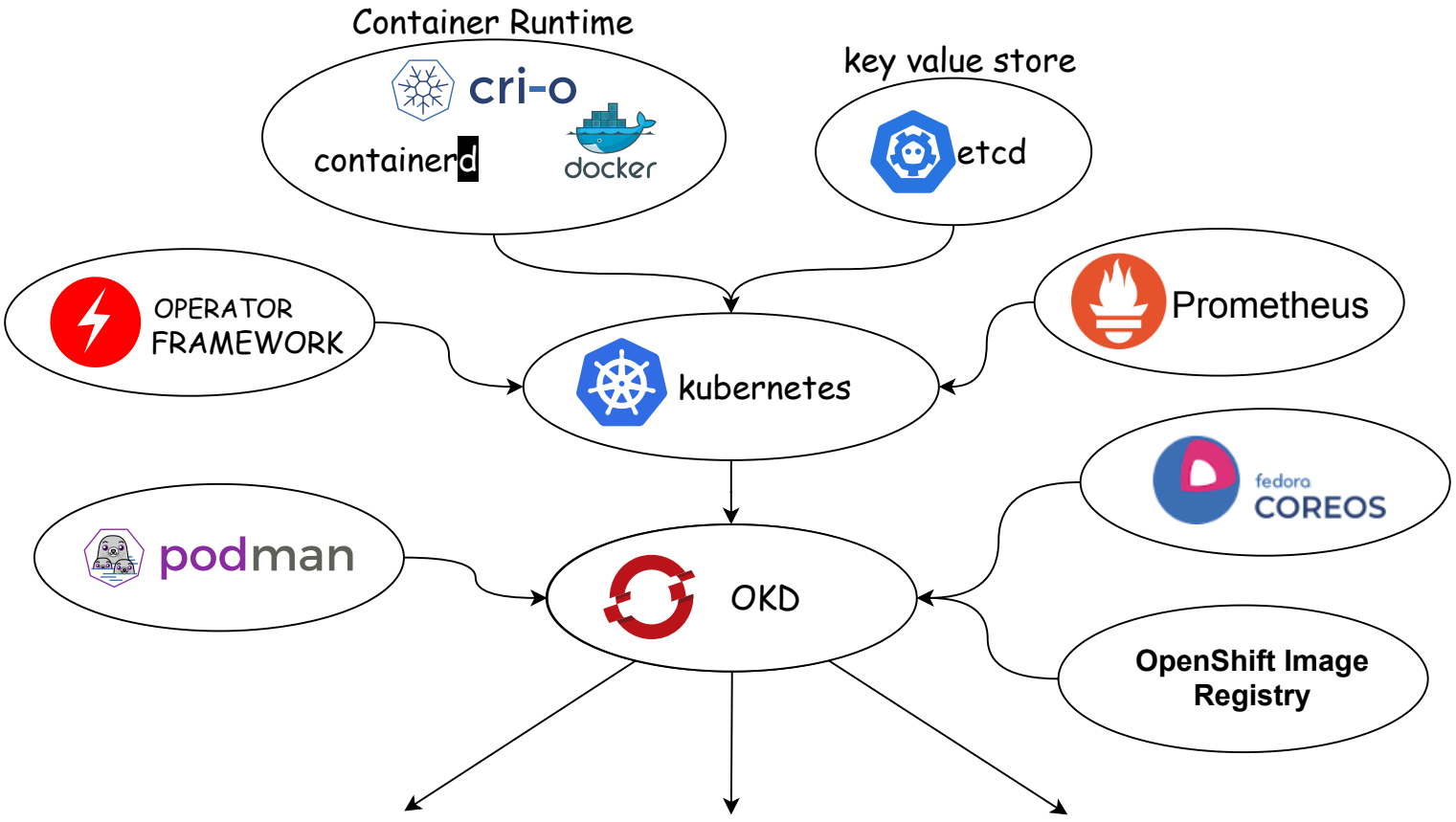



**Red Hat OpenShift Container Platform**

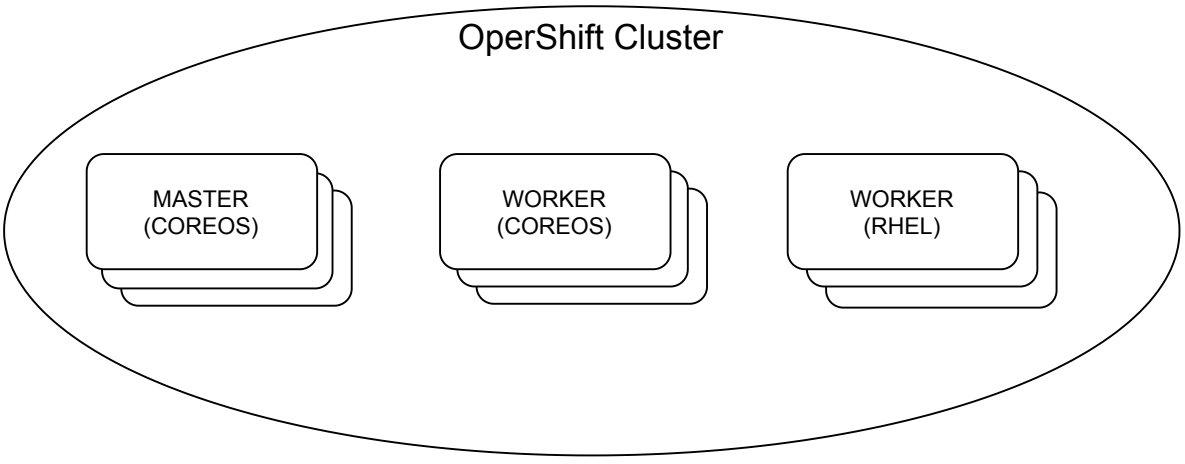
- Public/private DC.
- Bare metal and multiple cloud and virtualization providers.
- Full control by customer.

**Red Hat OpenShift Dedicated**

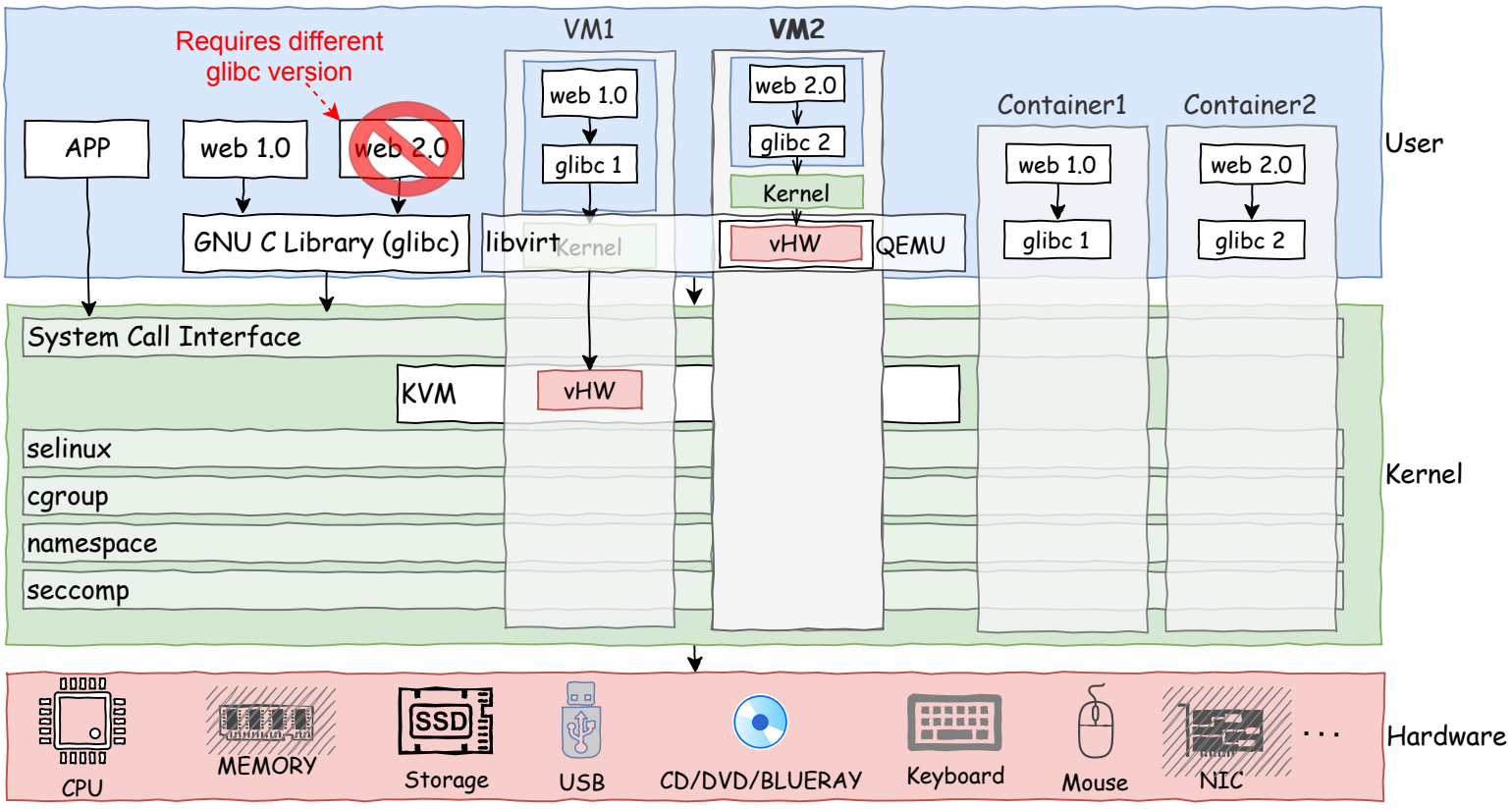
- Managed cluster in public cloud.
- RH manages the cluster.
- Customer manages updates and add-on services.

**Red Hat OpenShift Online**

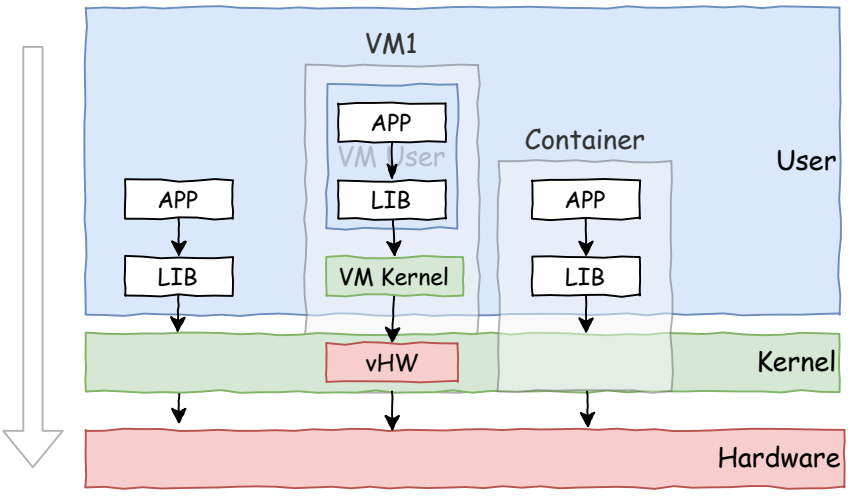
- Public hosted cluster.
- Shared resources by multiple customers.
- RH manages cluster life cycle.



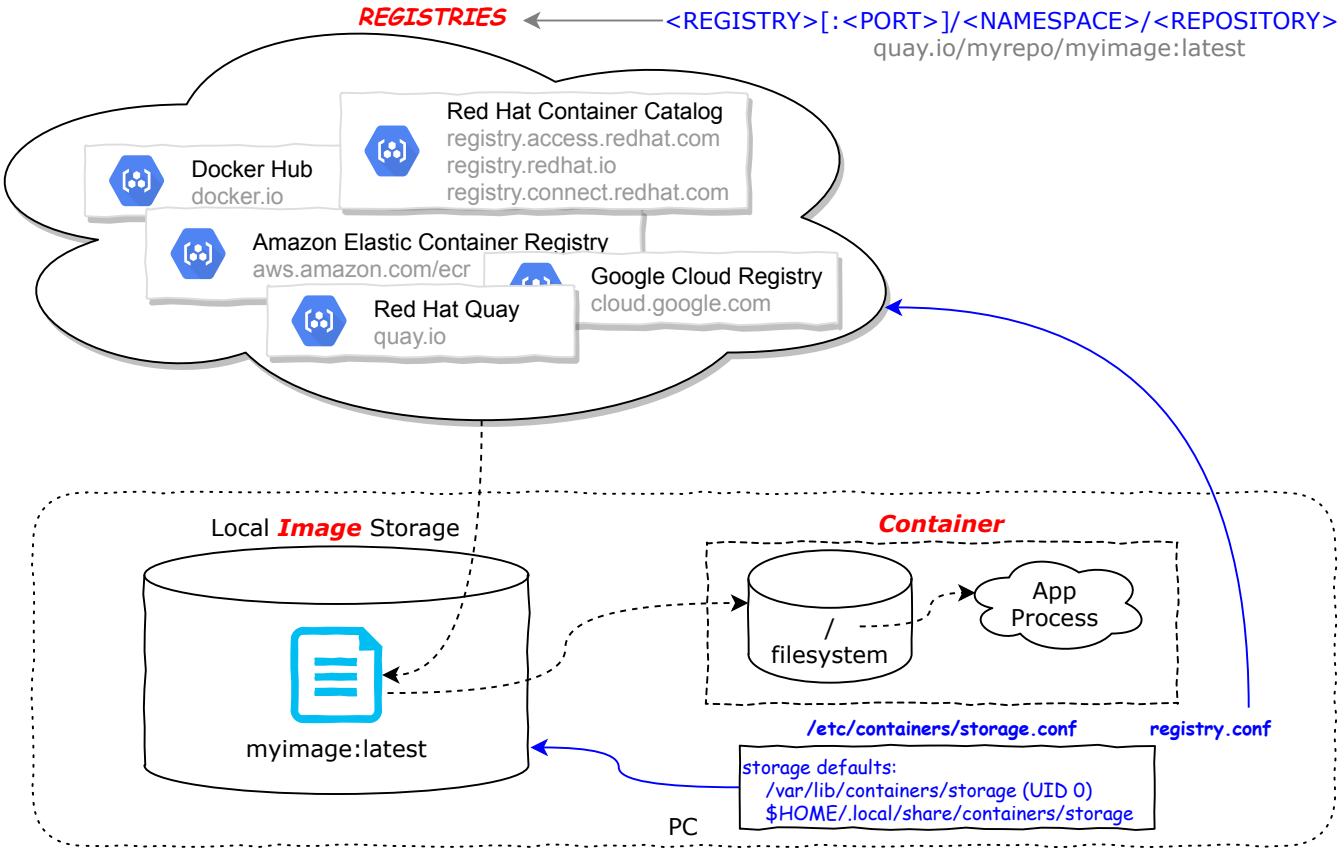
# VM vs Container

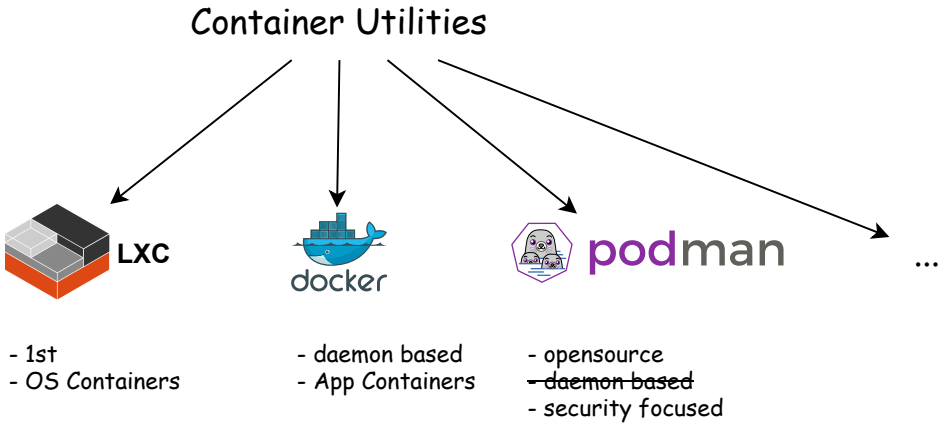



Ref: https://www.redhat.com/en/blog/all-you-need-know-about-kvm-userspace
https://www.packetcoders.io/what-is-the-difference-between-qemu-and-kvm/



# Container Architecture

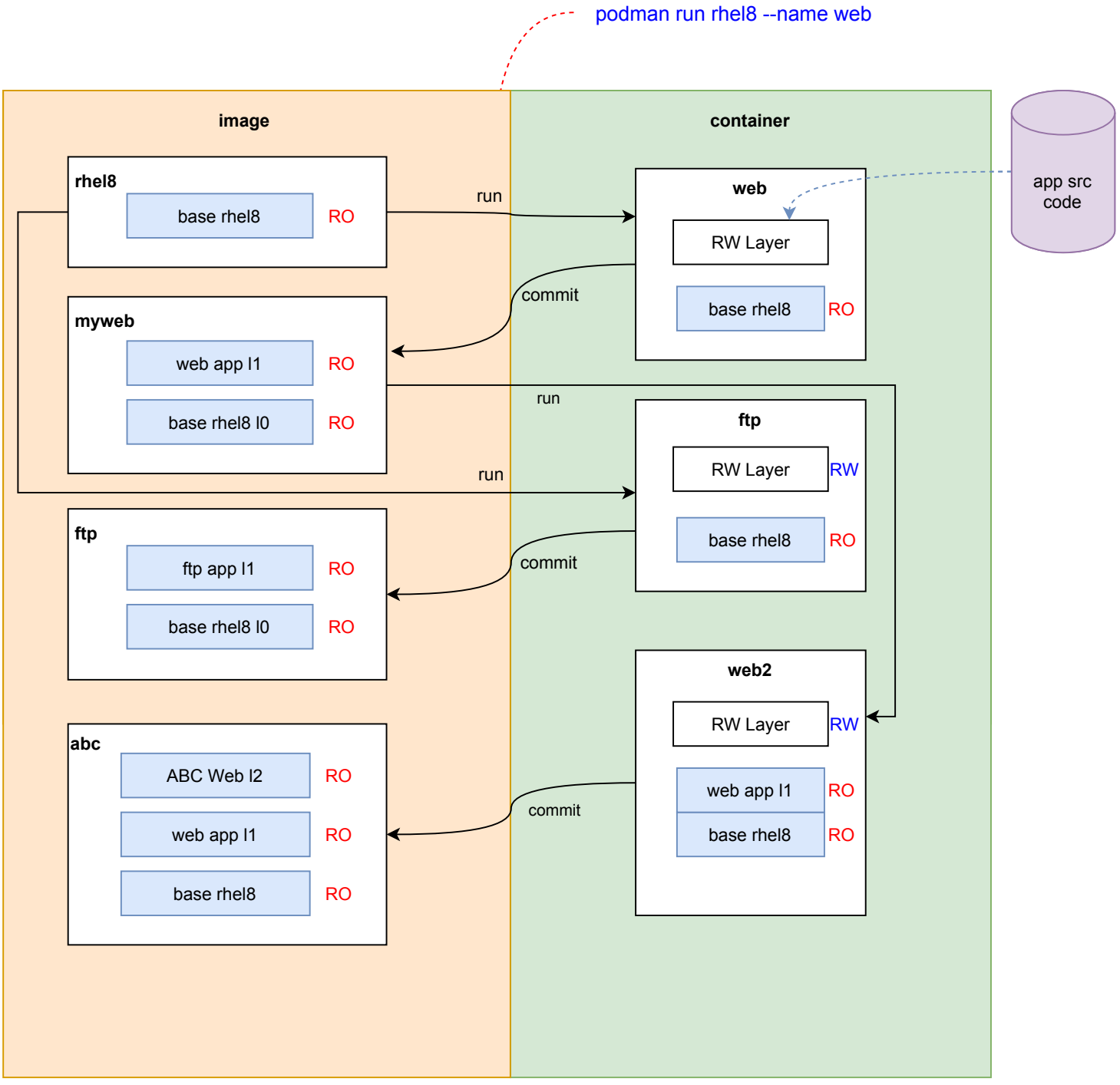


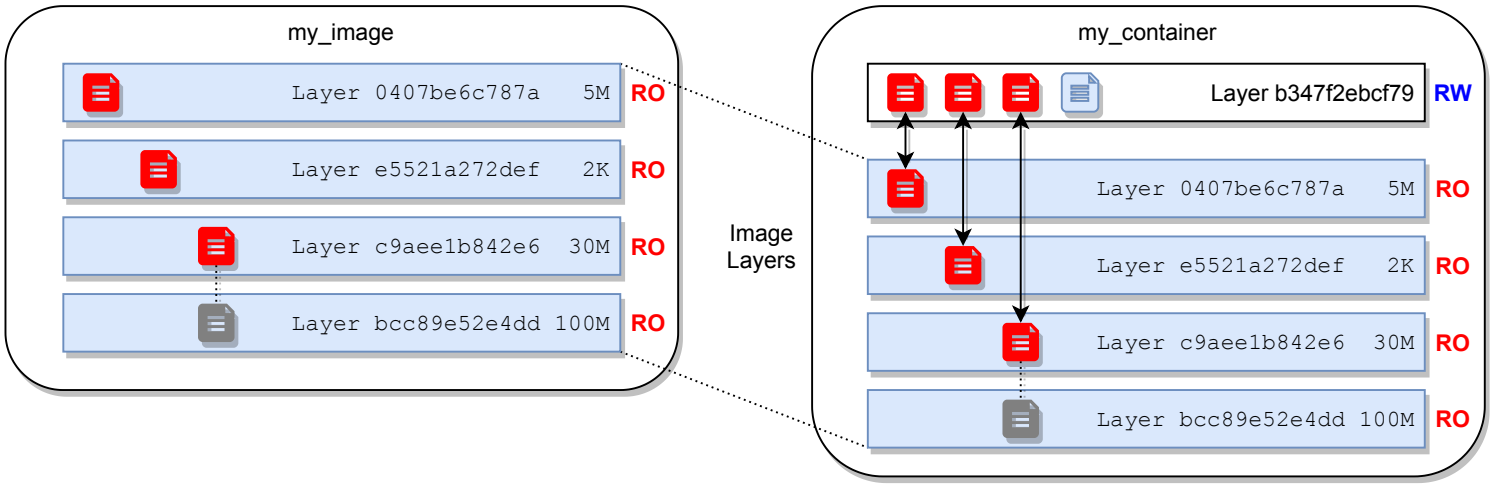

OS Container Vs Application Containers

# 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

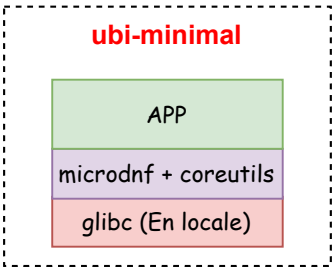


# UnionFS - A Stackable Unification File System




## BASE IMAGE TYPES

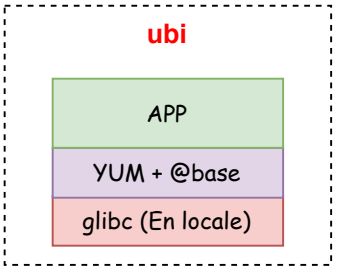
### MINIMAL




**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)

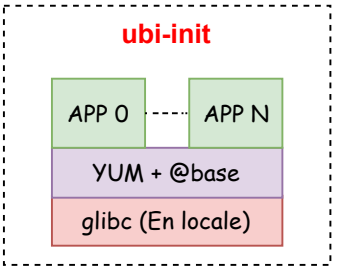
### PLATFORM




**For any apps that runs on RHEL**

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

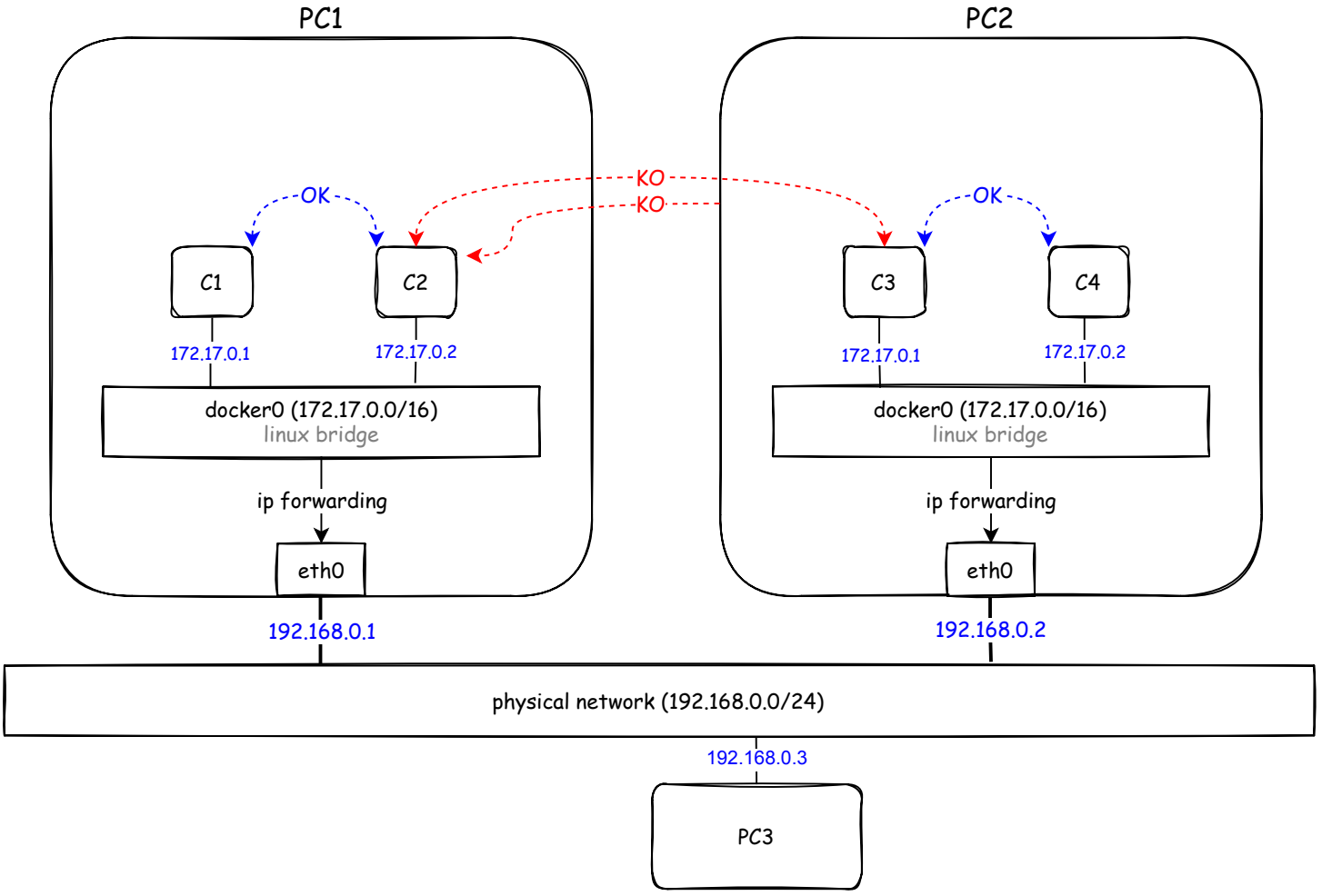
### MULTI-SERVICE

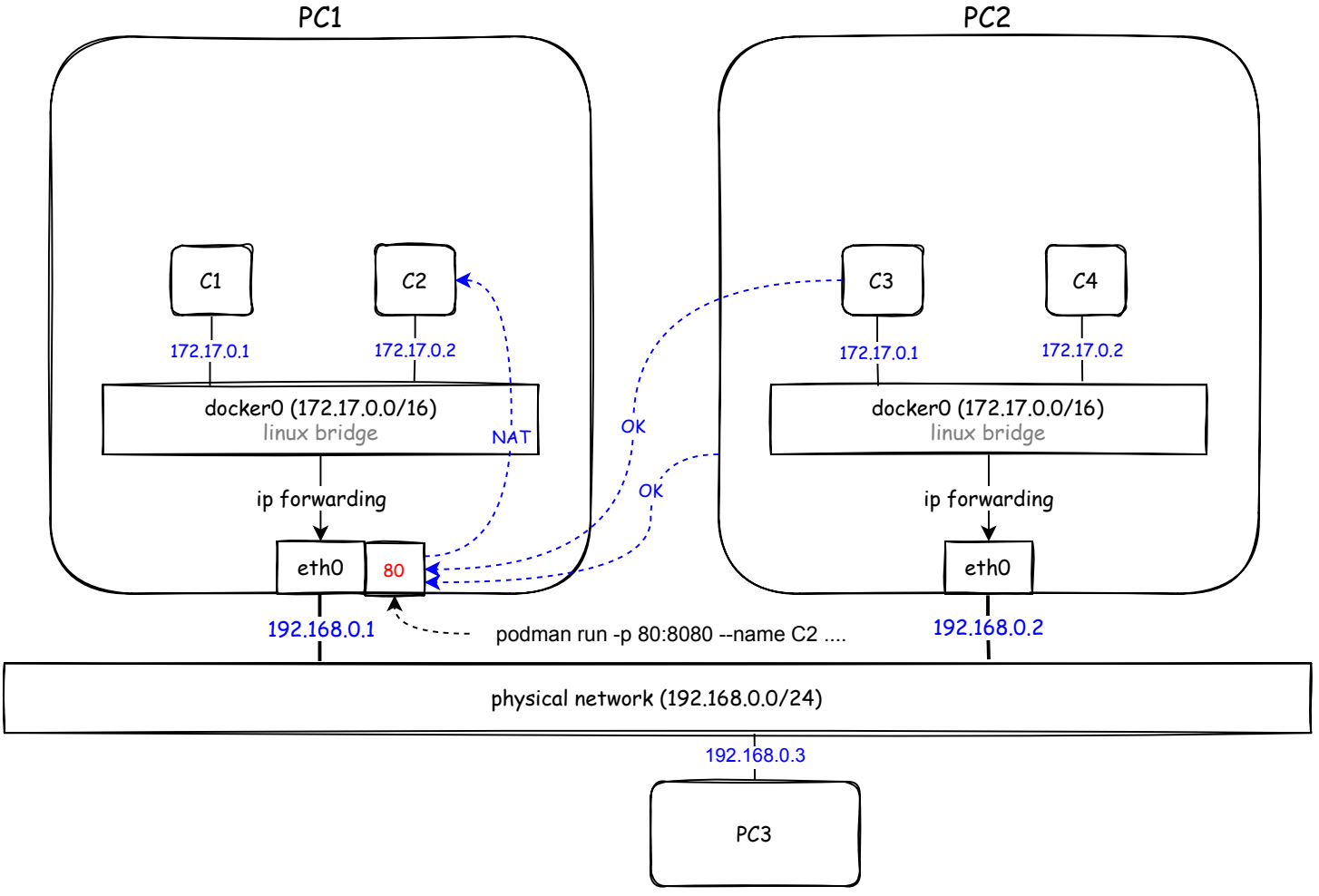


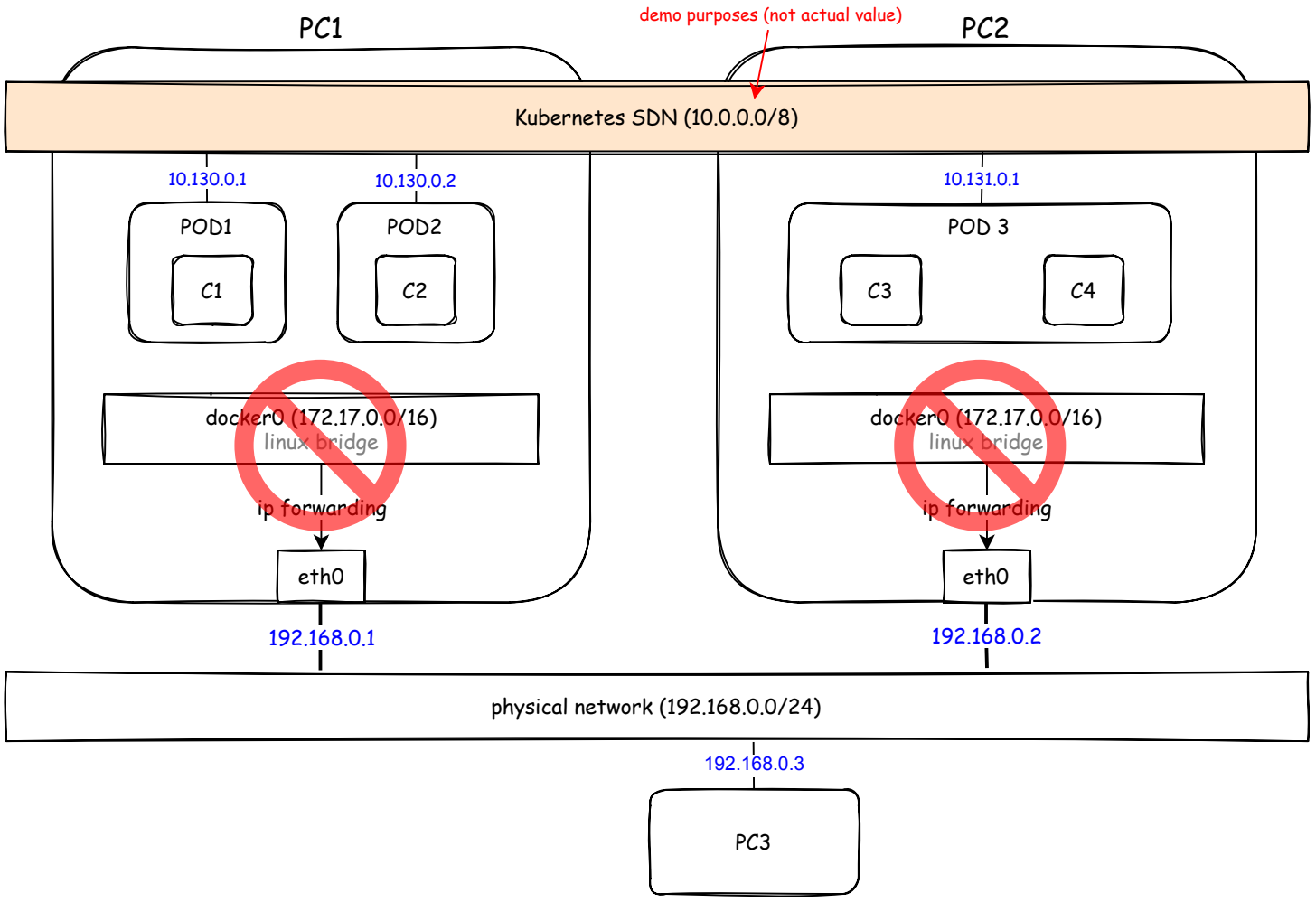

**Eases running multi-service in single container**

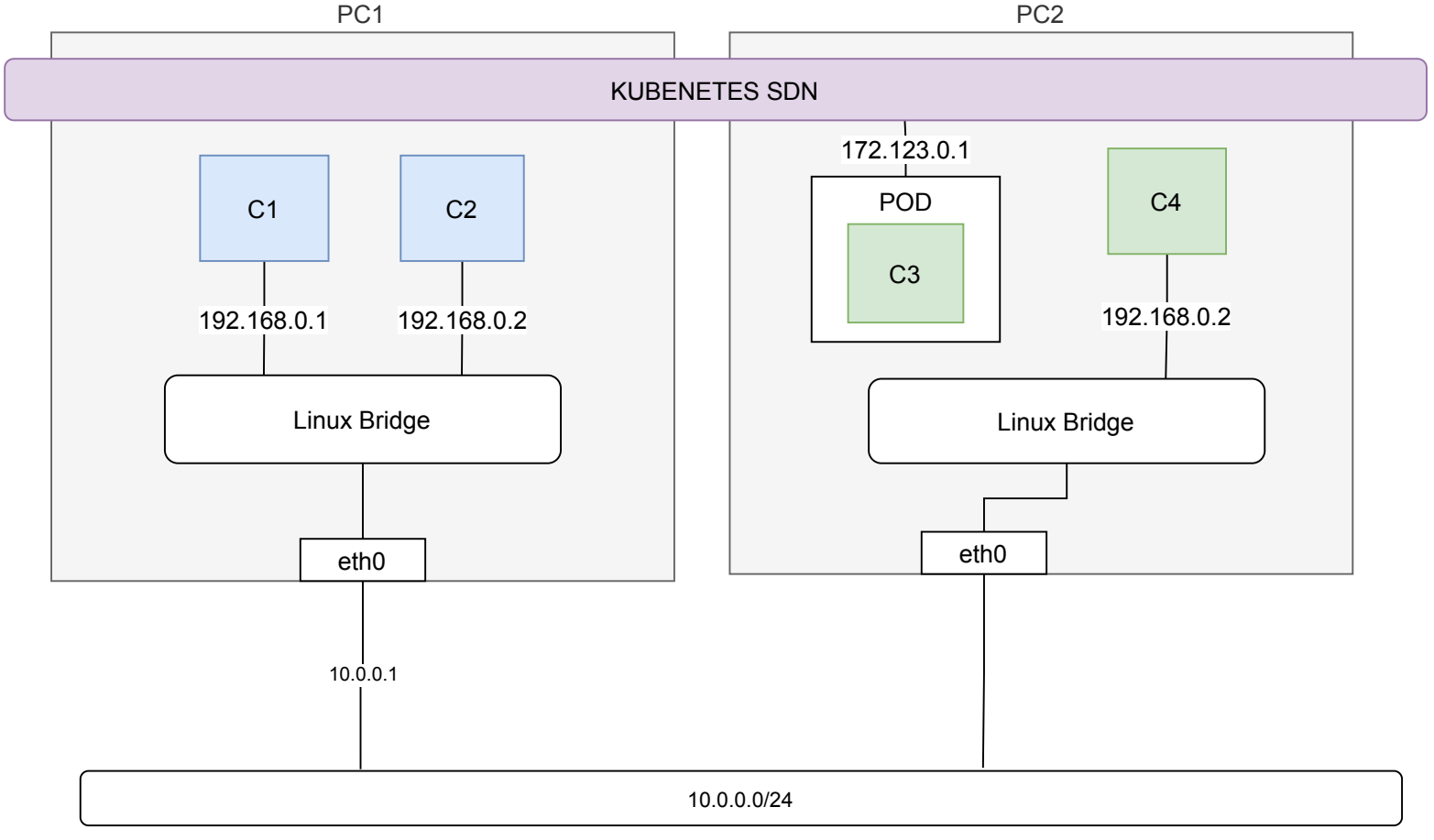
- configured to run systemd on start
- allows you to enable th services at build time

# Basic Network - Container vs Kubernetes

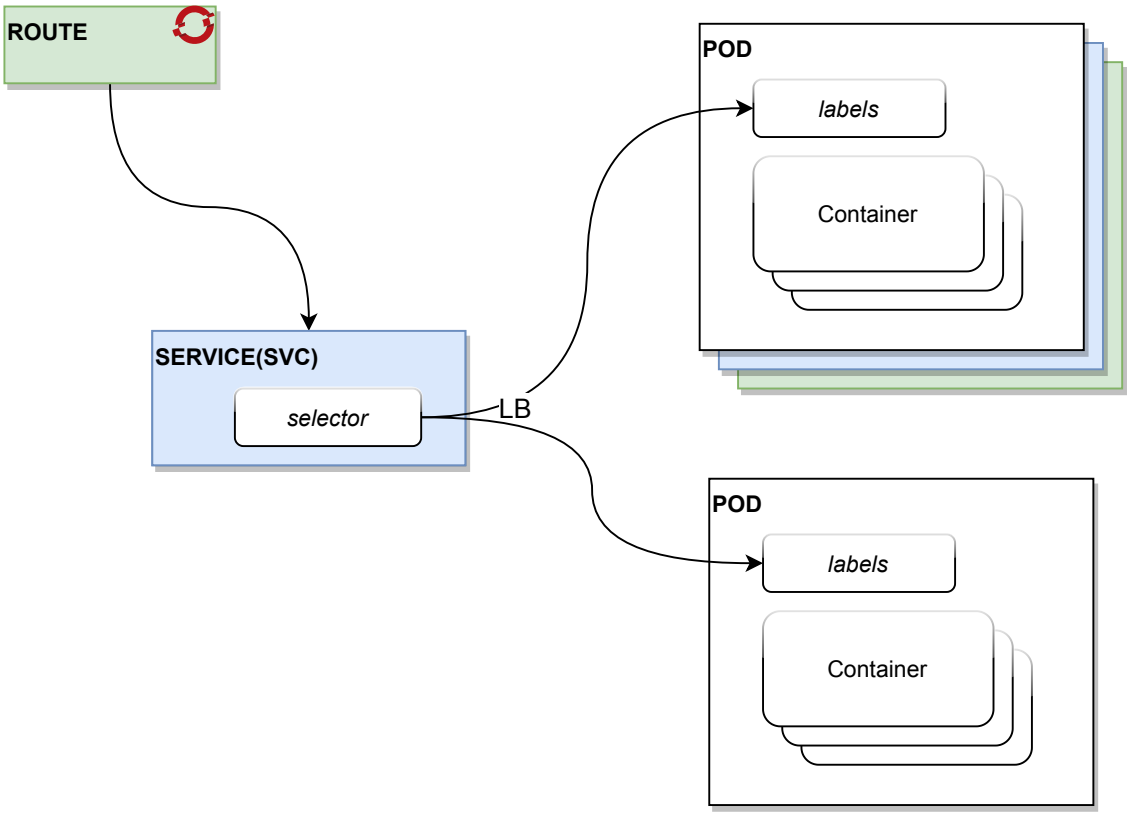



PC1
demo purposes (not actual value)
PC2
Kubernetes SDN (10.0.0.0/8)
10.130.0.1
10.130.0.2
10.131.0.1
POD1
C1
POD2
C2
POD 3
C3
C4
docker0 (172.17.0.0/16)
linux bridge
ip forwarding
eth0
192.168.0.1
192.168.0.2
physical network (192.168.0.0/24)
192.168.0.3
PC3


PC1
PC2
KUBENETES SDN
C1
C2
172.123.0.1
POD
C3
C4
192.168.0.1
192.168.0.2
192.168.0.2
Linux Bridge
Linux Bridge
eth0
eth0
10.0.0.1
10.0.0.0/24

# Route, Service and Pod Relationship



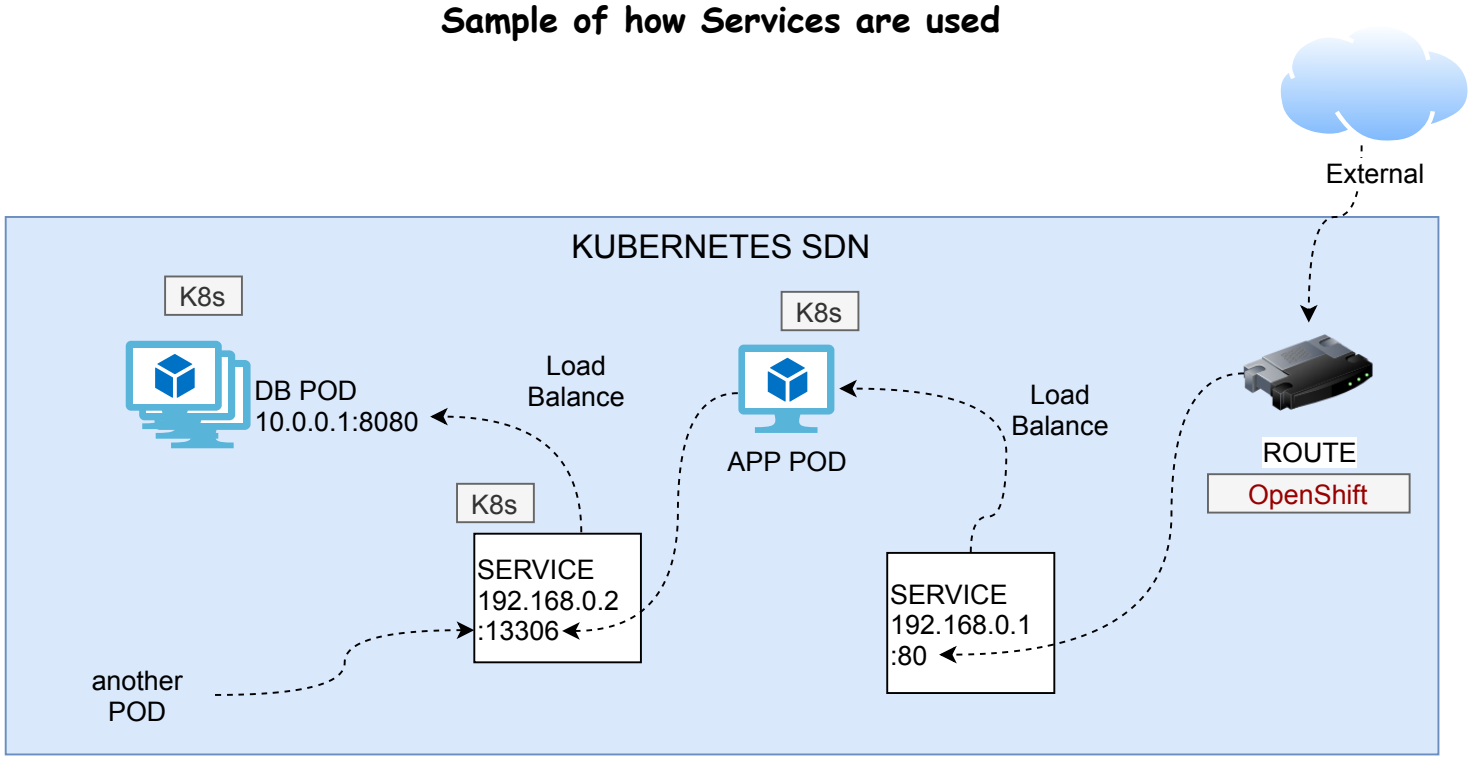

**POD**
A pod contains one or more containers.

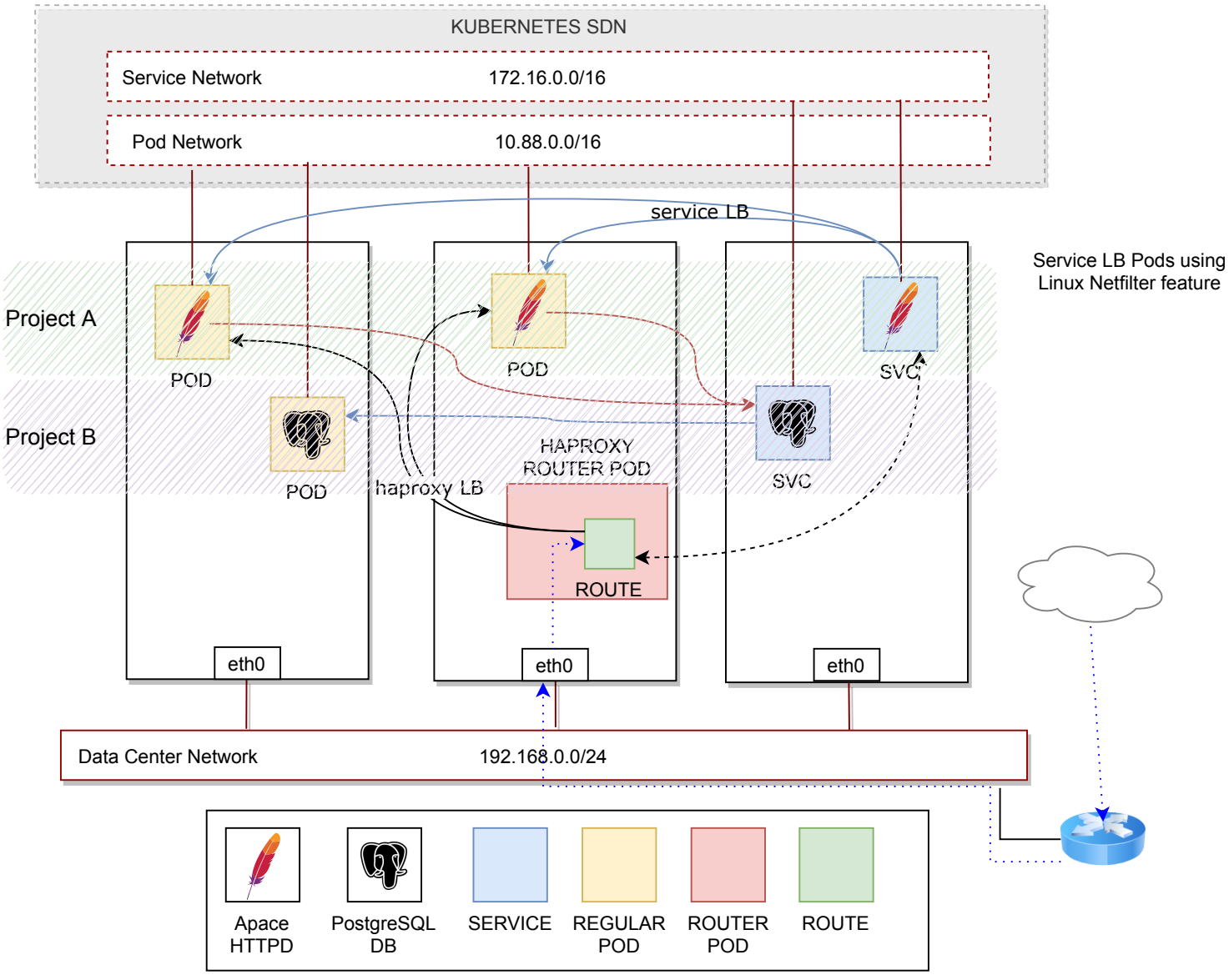
**SERVICE**
A service references the pod(s) by using the label selector.
The service load balances the connections between all the pods.

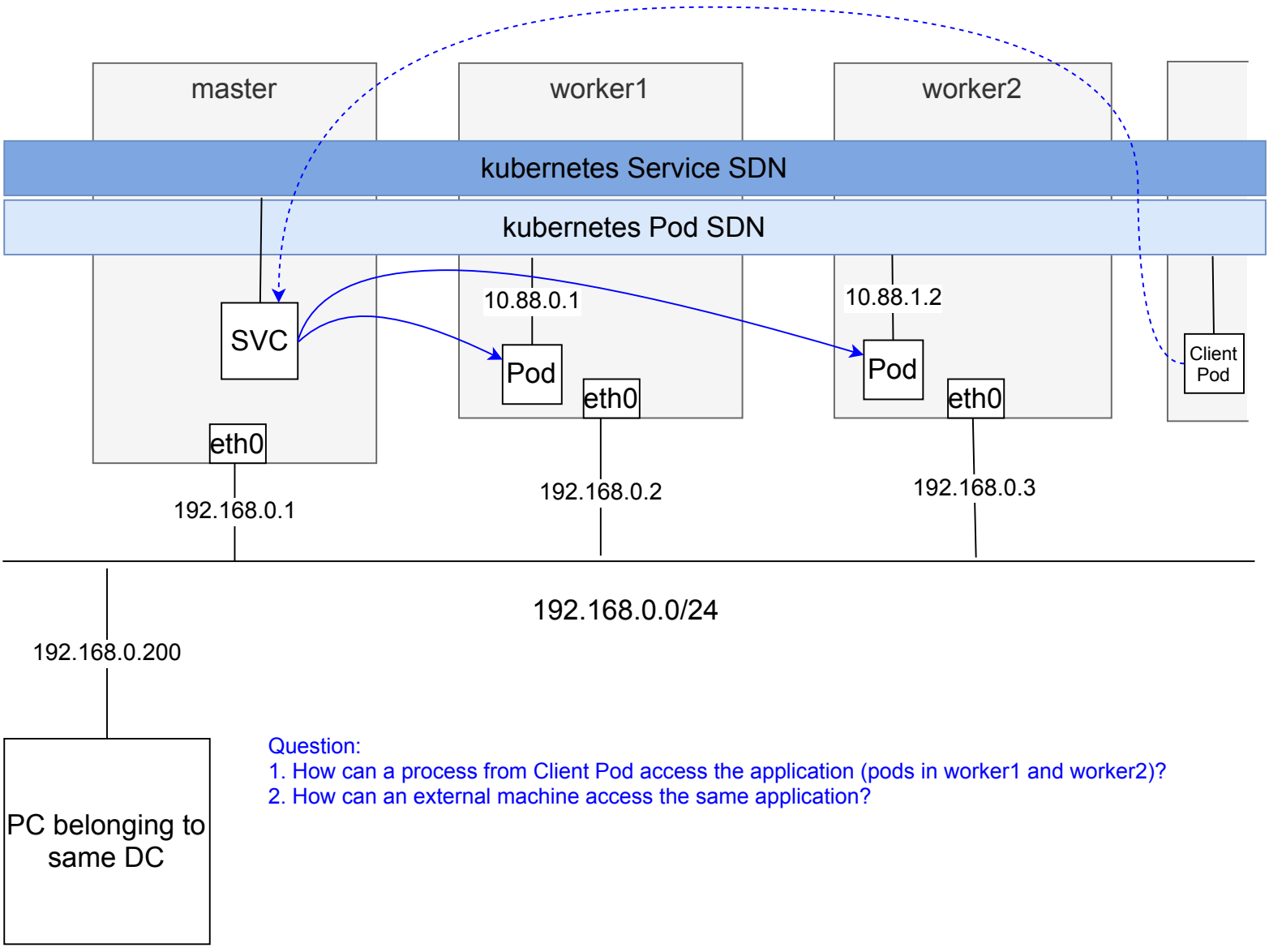
**ROUTE**
A route exposes the service to the external world.

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

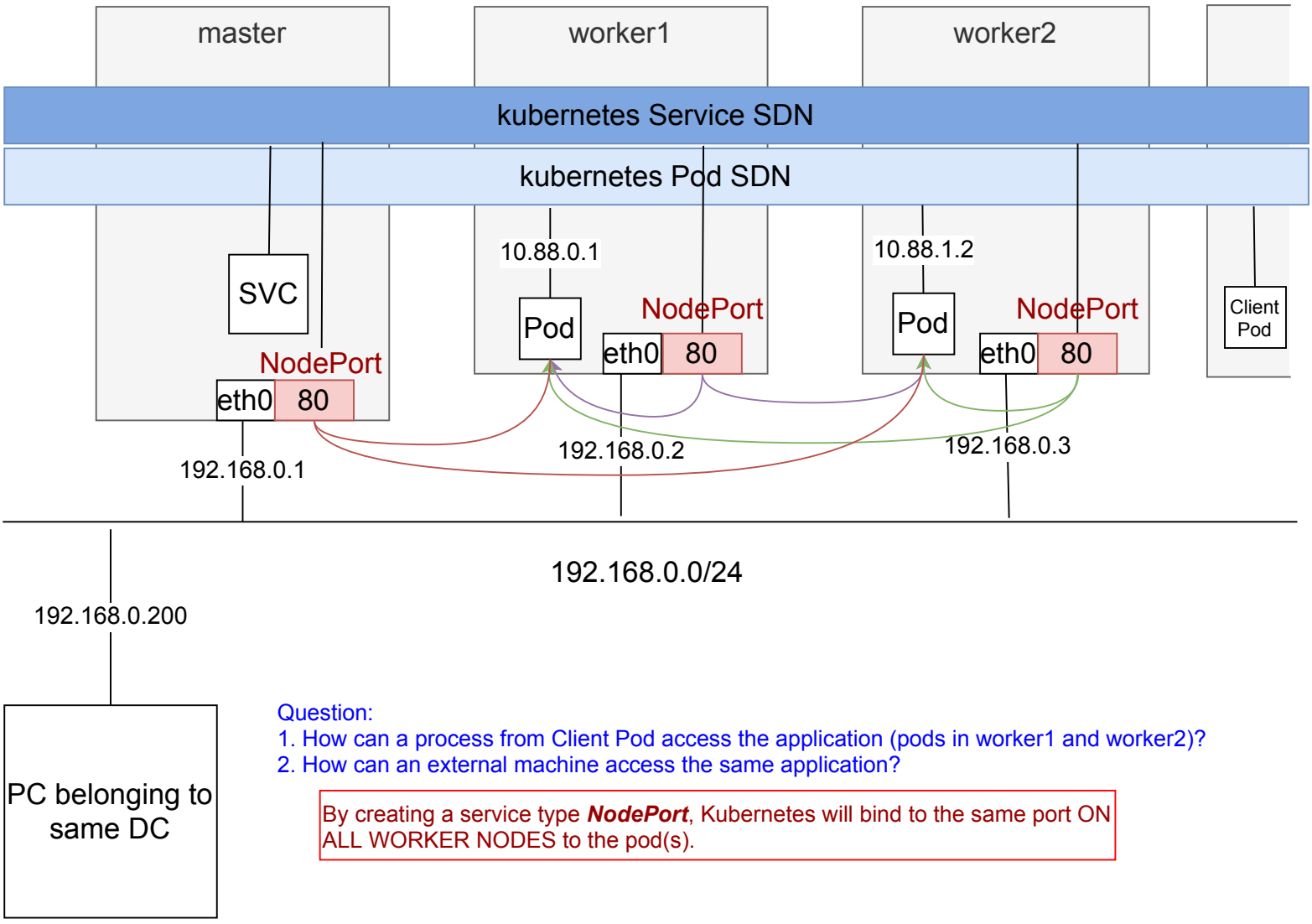
# Sample of how Services are used





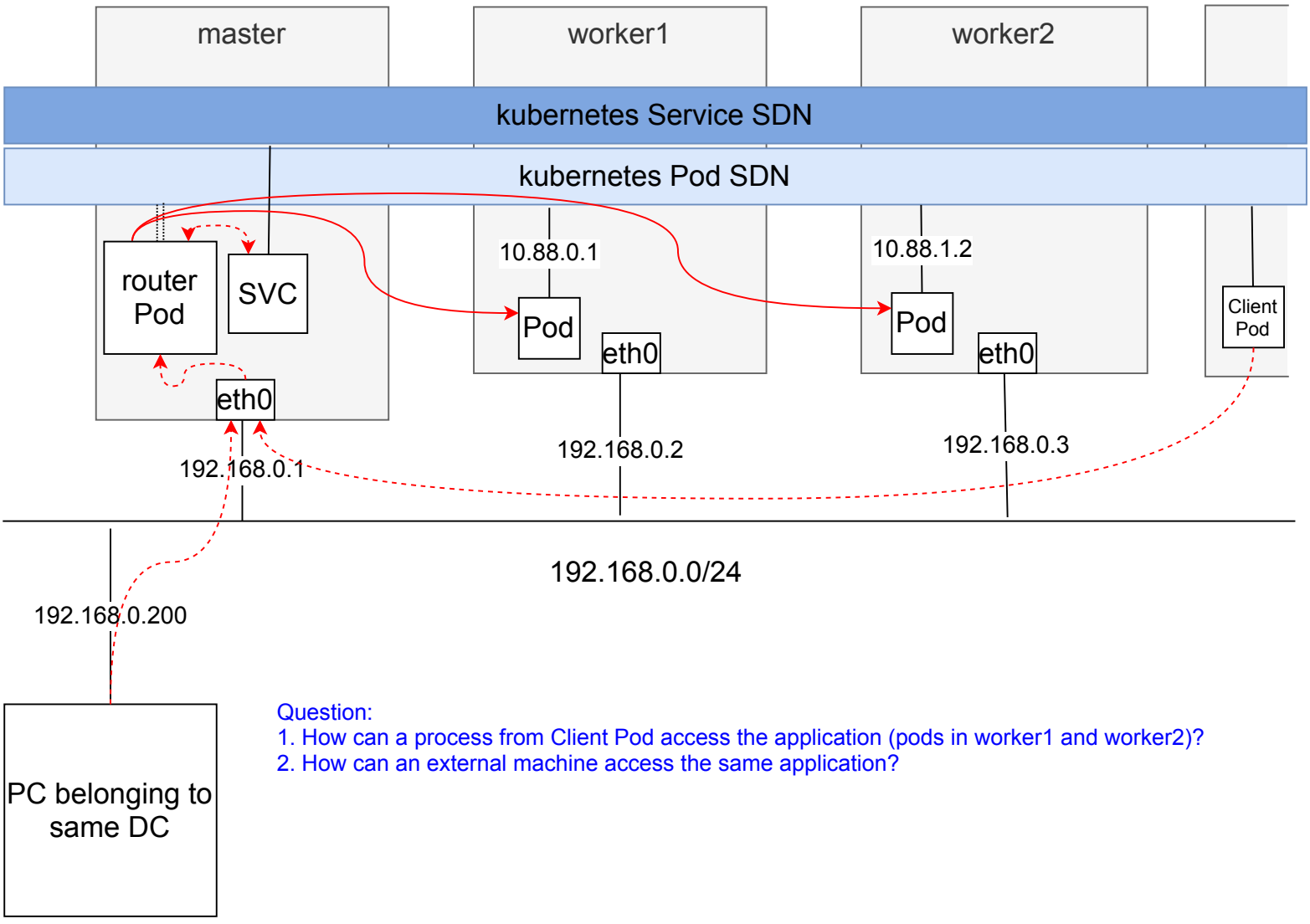

Question:
1. How can a process from Client Pod access the application (pods in worker1 and worker2)?
2. How can an external machine access the same application?




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?

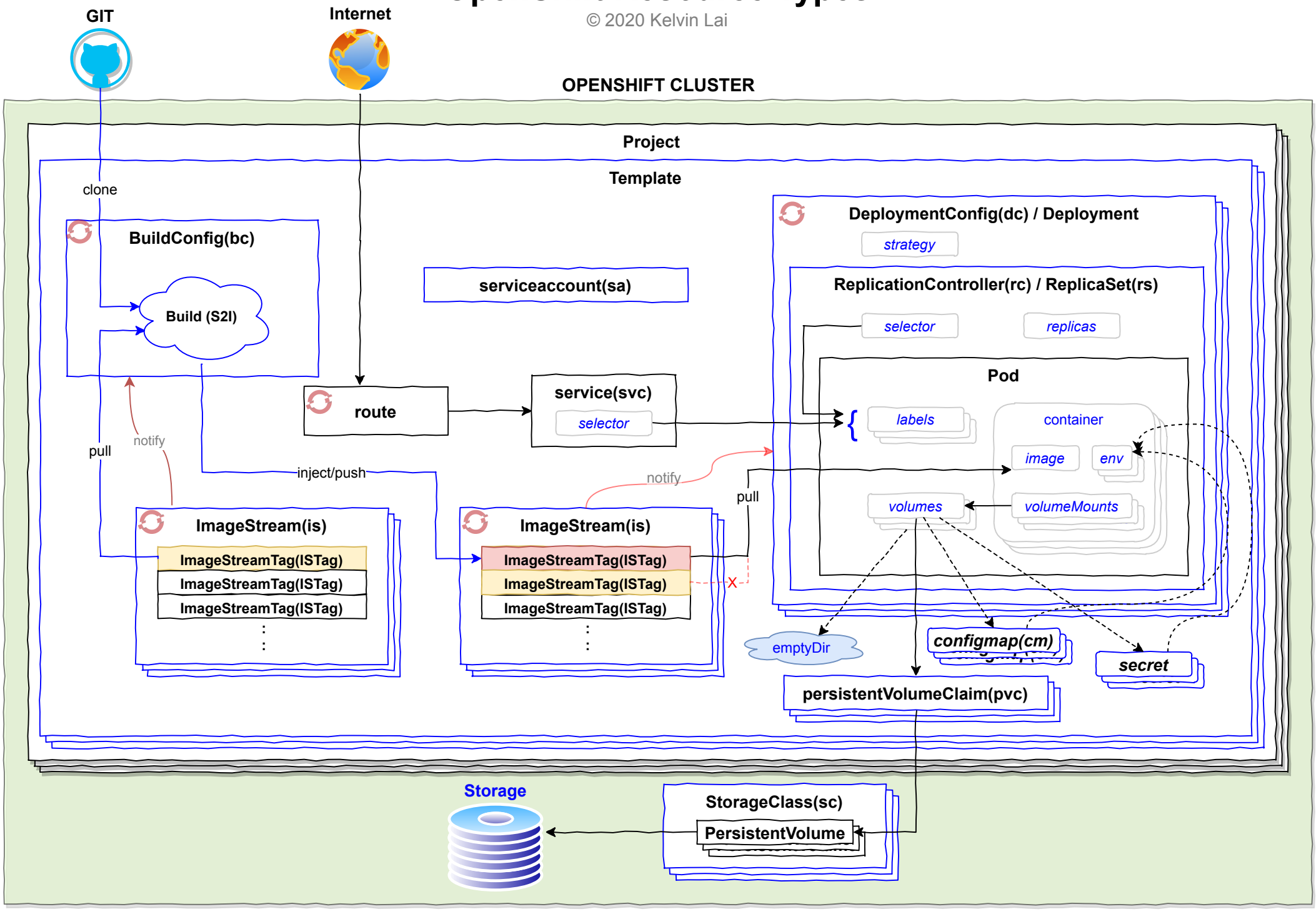
By creating a service type ***NodePort***, Kubernetes will bind to the same port ON ALL WORKER NODES to the pod(s).




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?

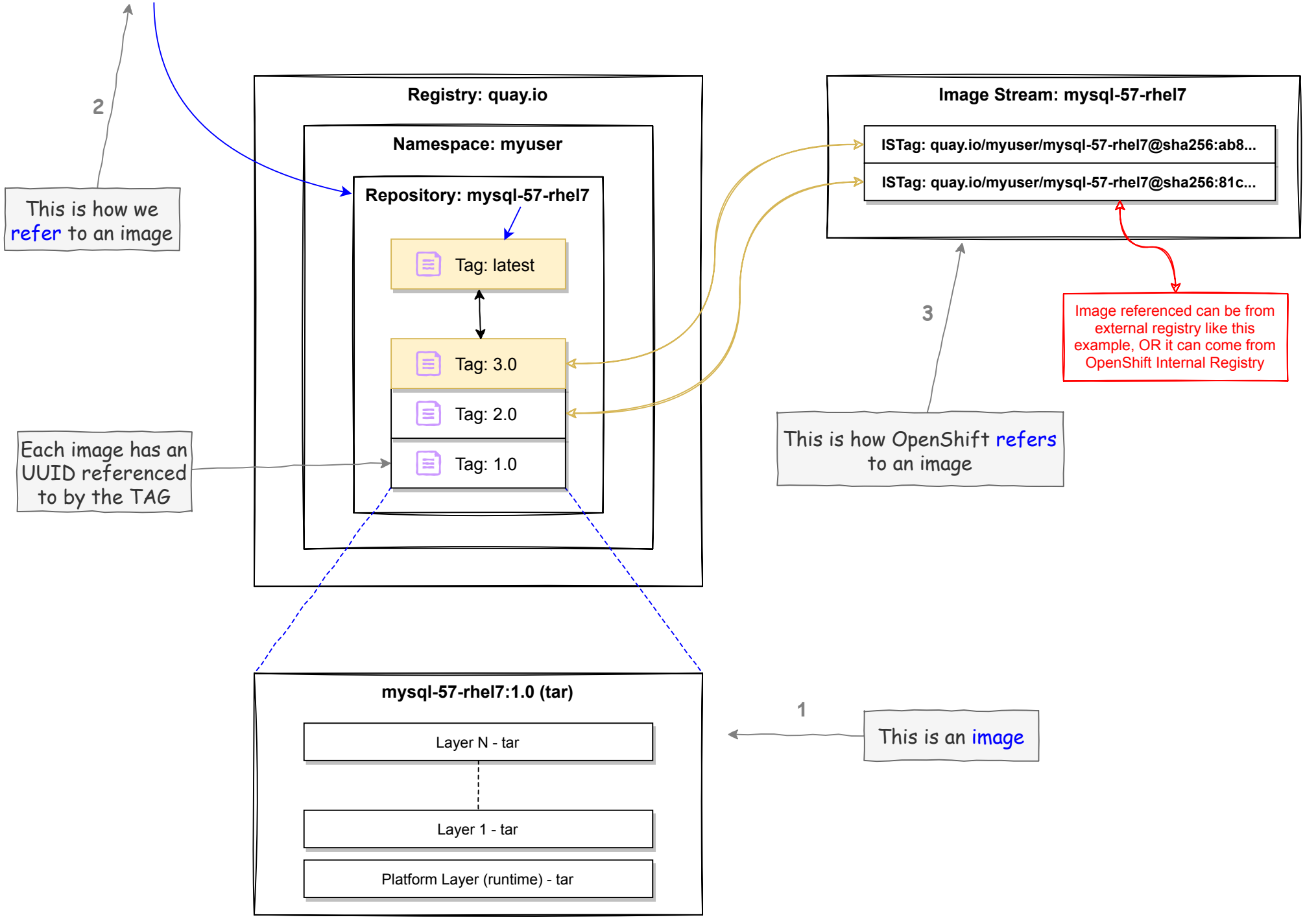
# OpenShift Resource Types





**Container Image Naming Convention: <REGISTRY>[:<PORT>]/<NAMESPACE>/<REPOSITORY>[:<TAG>]**

**podman pull quay.io/myuser/mysql-57-rhel7**



# 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>
```

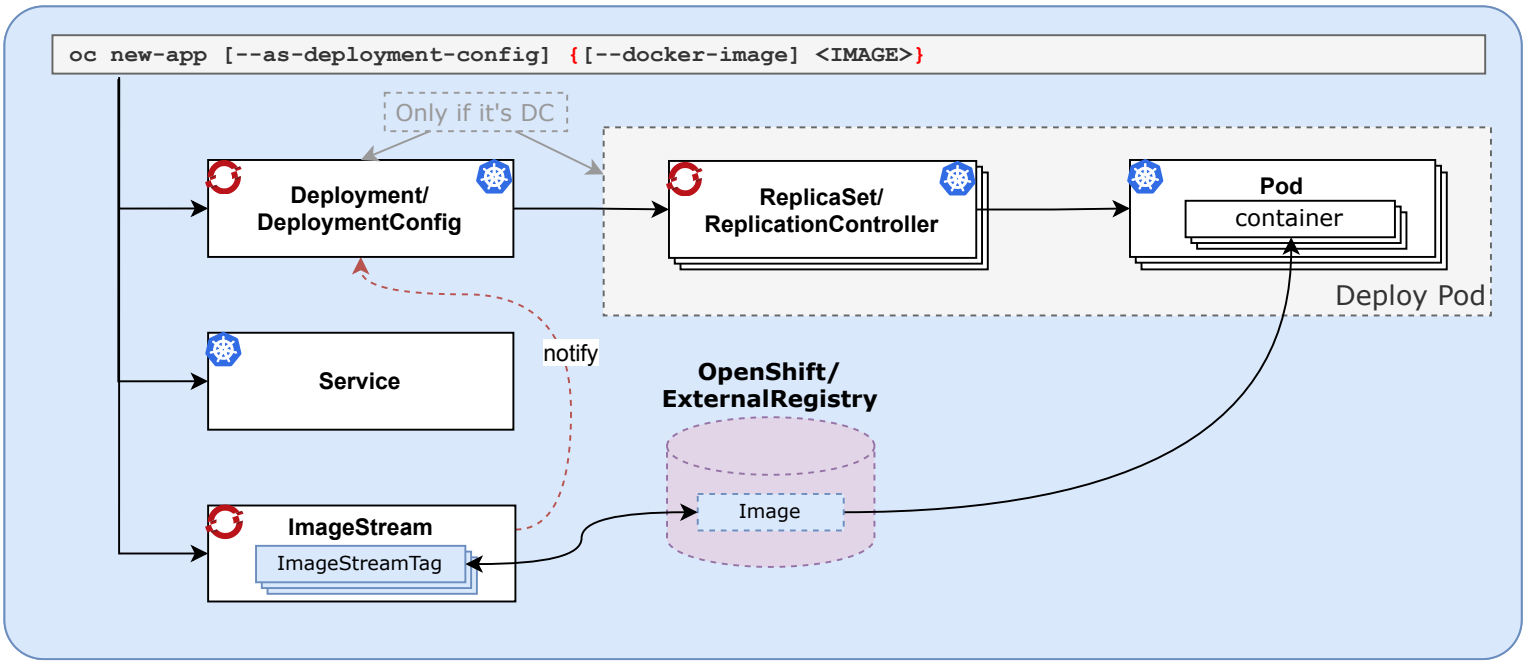
3. Using yaml/json file

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

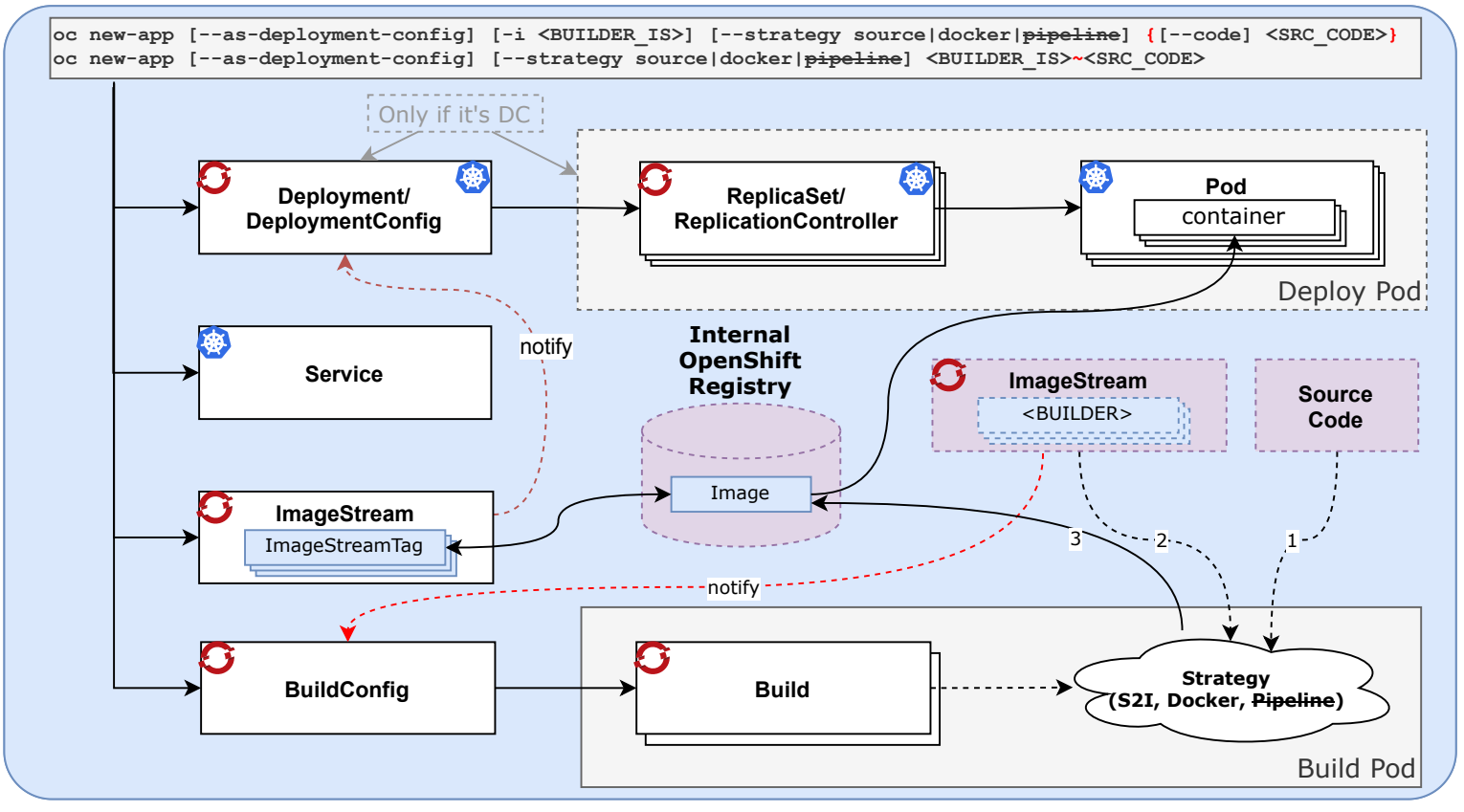
4. Using template

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

### 1. Use Existing Image




### 2. Managed Life Cycle



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

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

```
oc new-app php: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.

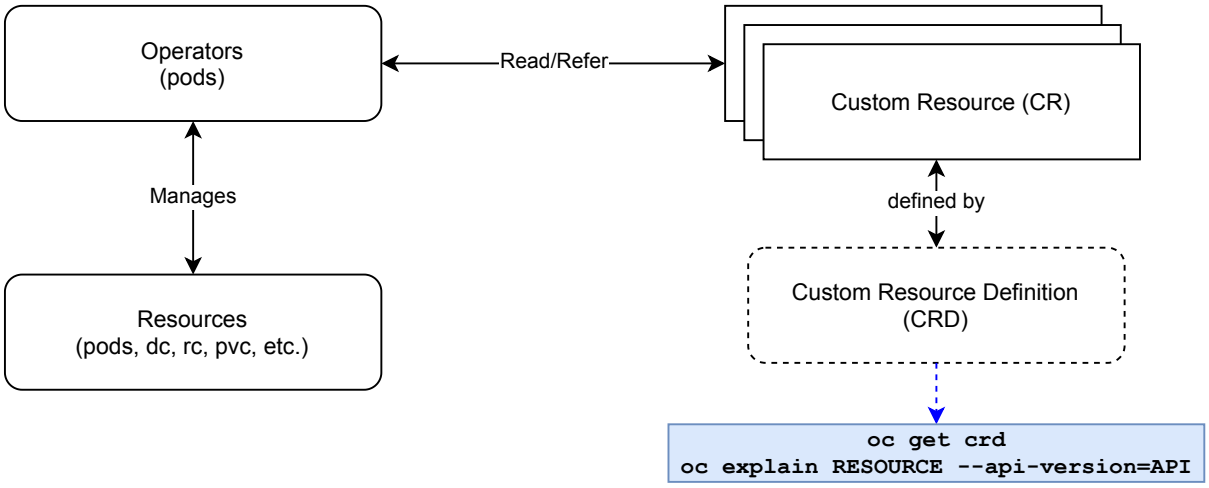

**oc expose <DC/DEPLOYMENT/RC/RS/POD> <RESOURCE_NAME>**

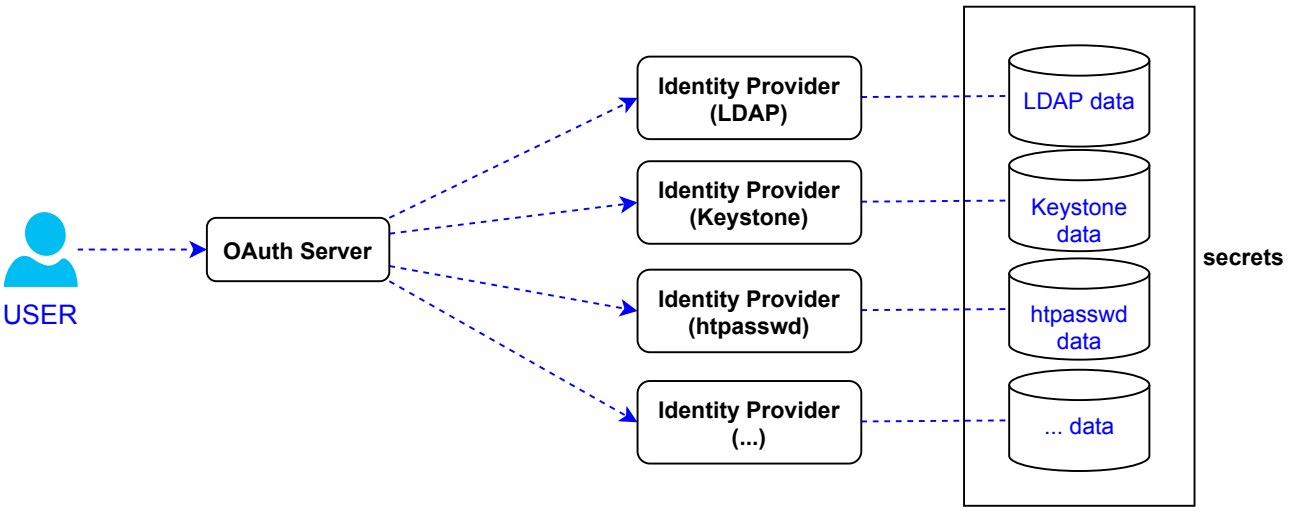
**DNS NAME = <SVC>.<PROJ>[.svc.cluster.local]**
**ENVIRONMENT VARIABLE IN POD = <SVC>_SERVICE_HOST**


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

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

# Operators


Operators
(pods)
Read/Refer
Custom Resource (CR)
Manages
defined by
Resources
(pods, dc, rc, pvc, etc.)
Custom Resource Definition
(CRD)
oc get crd
oc explain RESOURCE --api-version=API




```
$ 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***

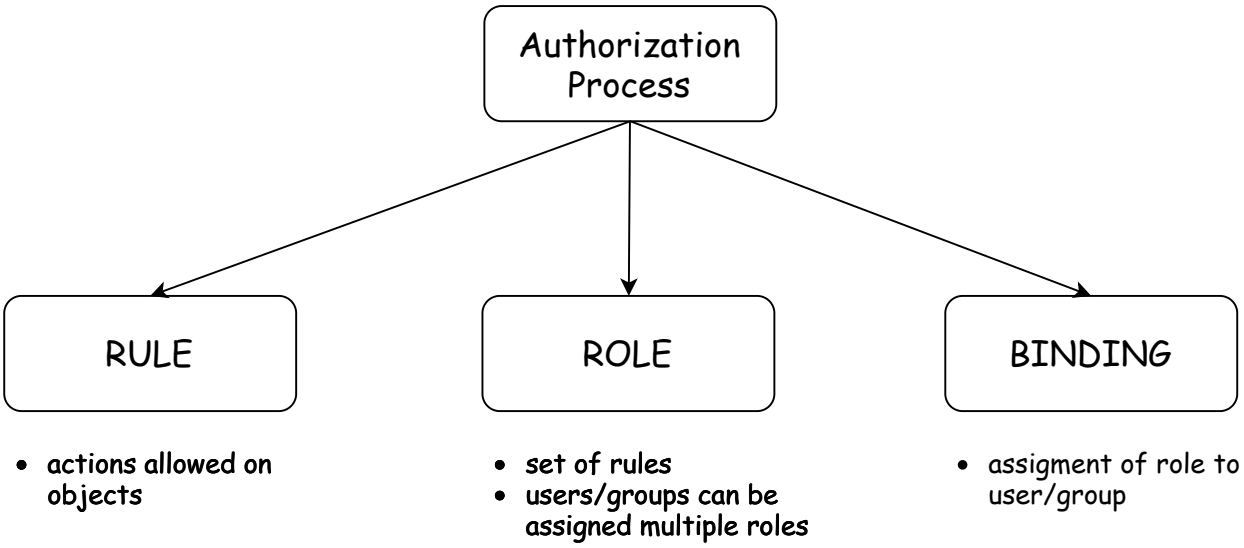
```
$ oc get pods -n openshift-authentication
```

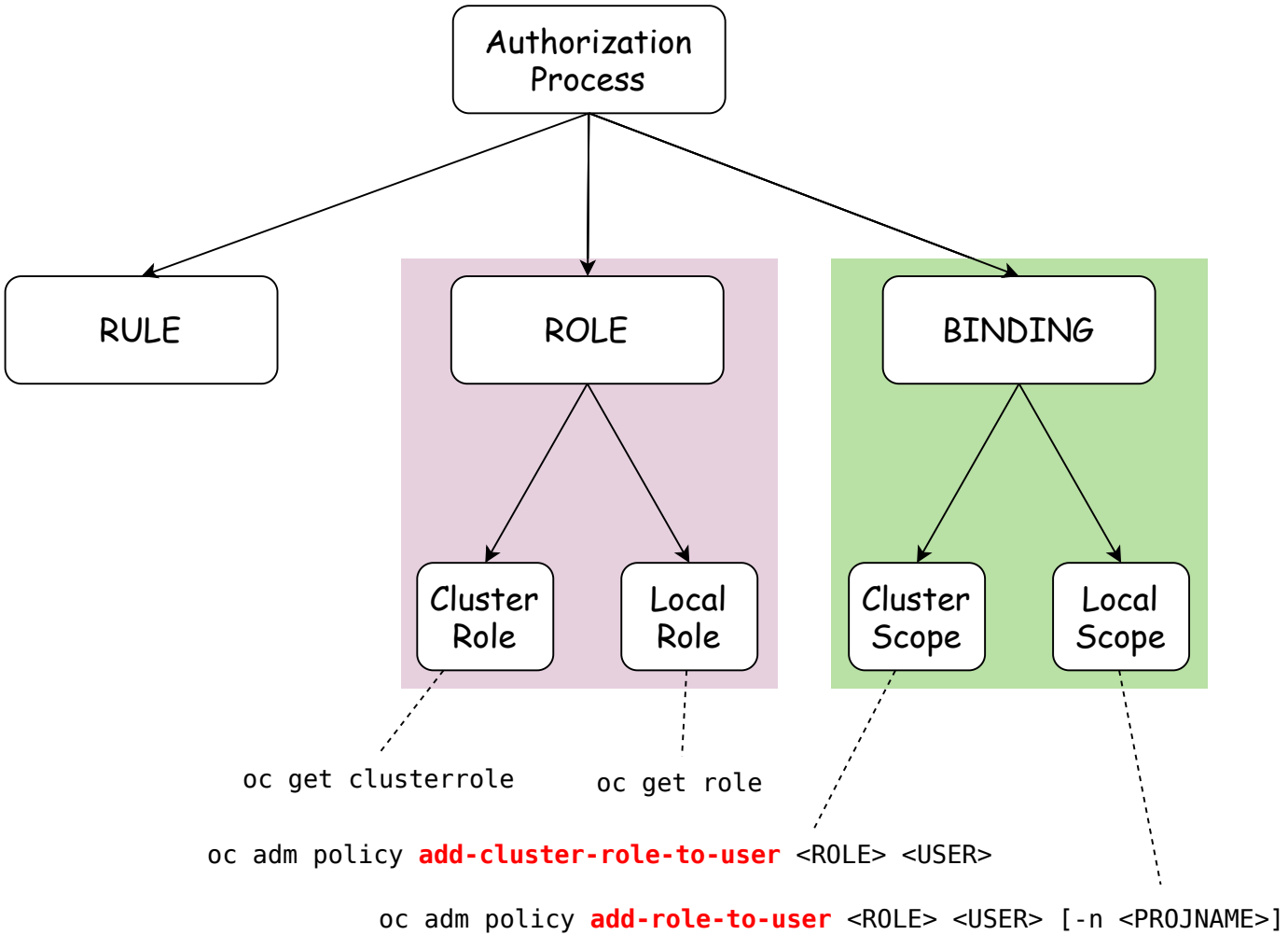
```
$ oc get oauth cluster -o yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
  ...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: htpasswd-secret
    mappingMethod: claim
    name: htpasswd_provider
    type: HTPasswd
```

**secret**

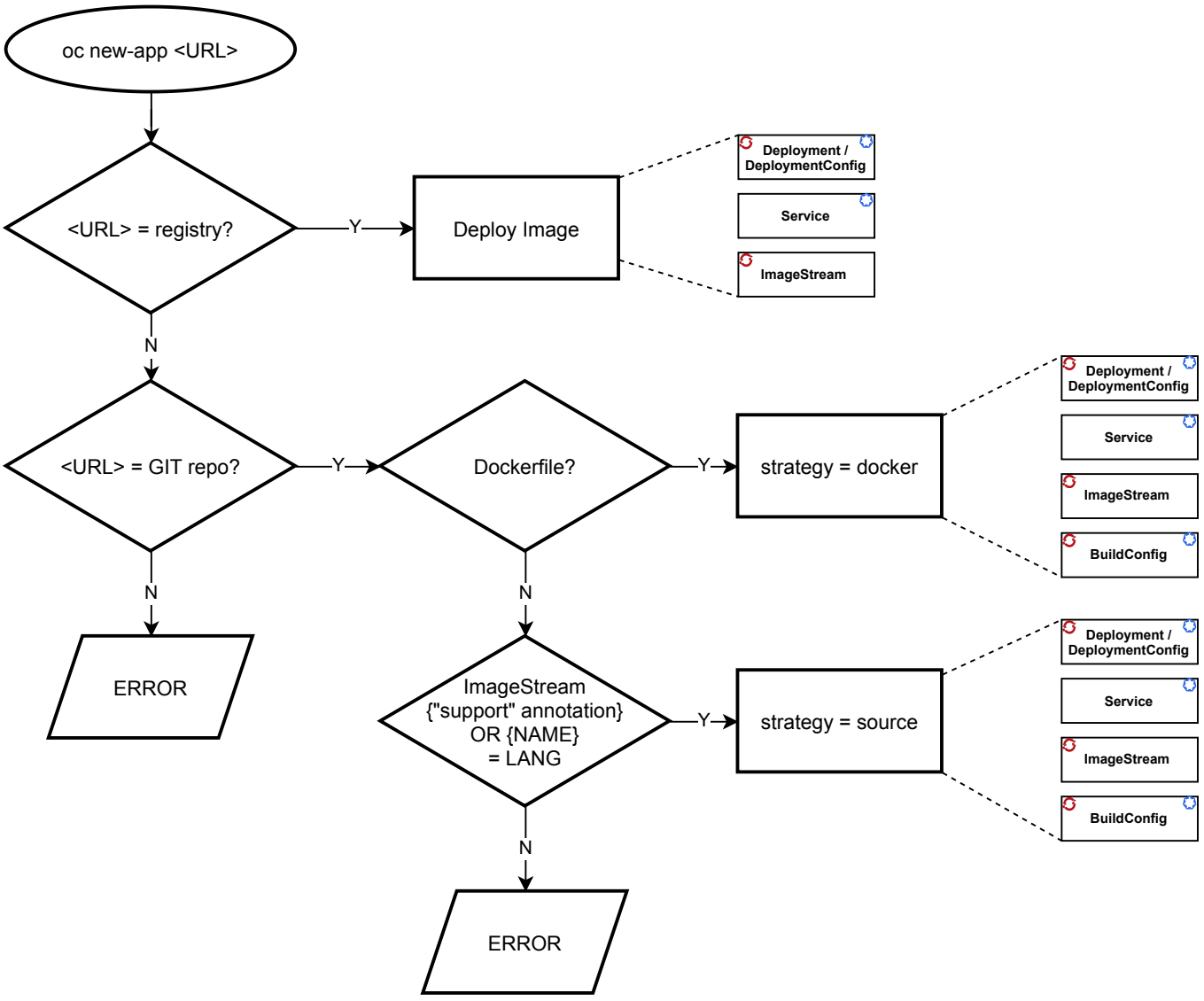
**Project: *openshift-config***

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


Authorization Process
RULE
ROLE
BINDING
actions allowed on objects
set of rules
users/groups can be assigned multiple roles
assigment of role to user/group

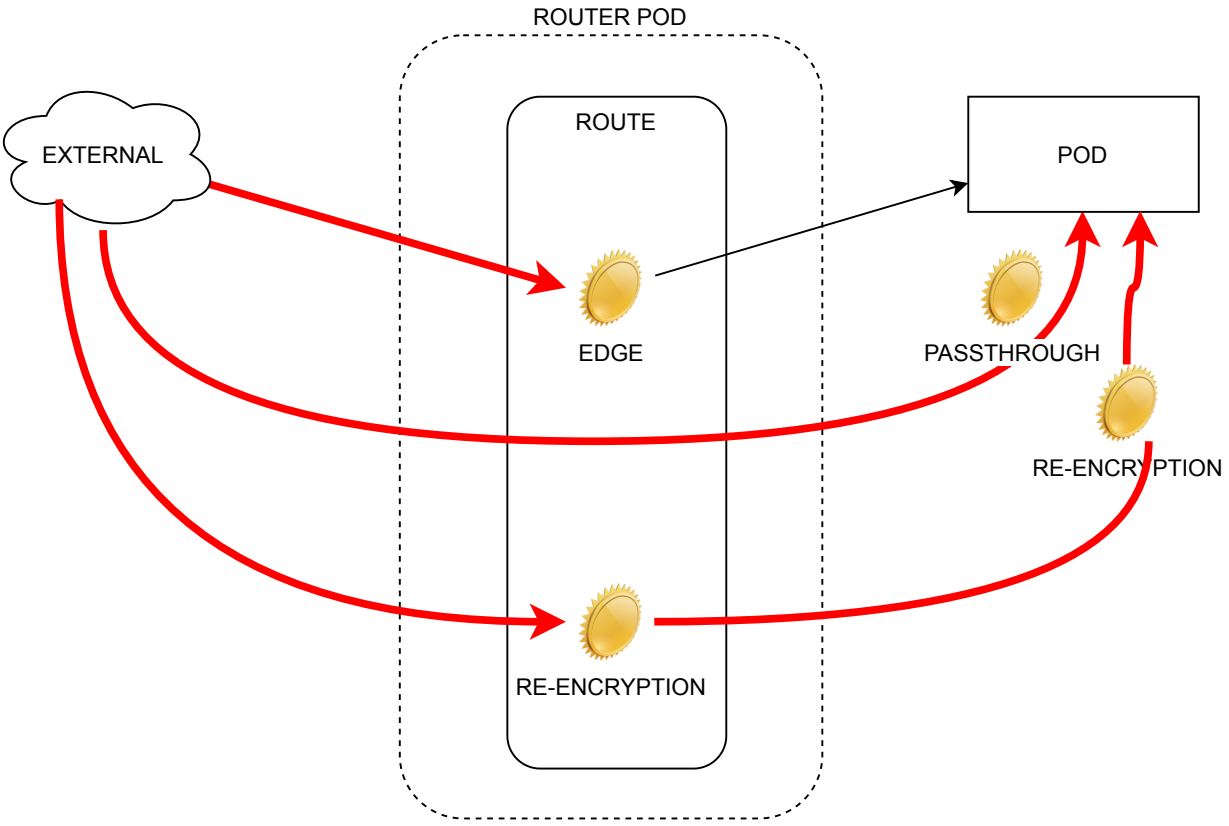

Authorization Process
RULE
ROLE
BINDING
Cluster Role
Local Role
Cluster Scope
Local Scope
oc get clusterrole
oc get role
oc adm policy add-cluster-role-to-user <ROLE> <USER>
oc adm policy add-role-to-user <ROLE> <USER> [-n <PROJNAME>]

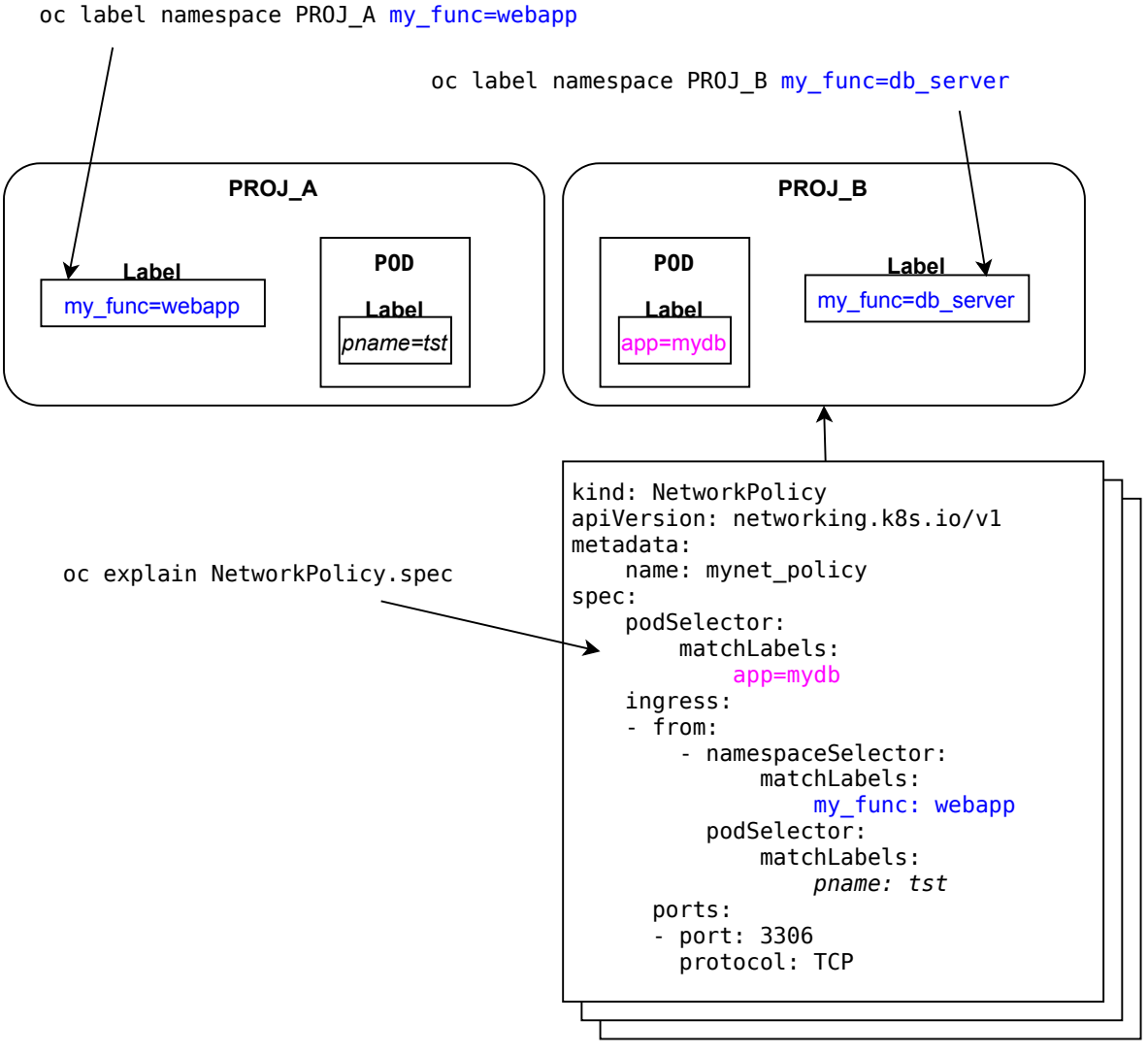
# oc new-app flowchart



# Template

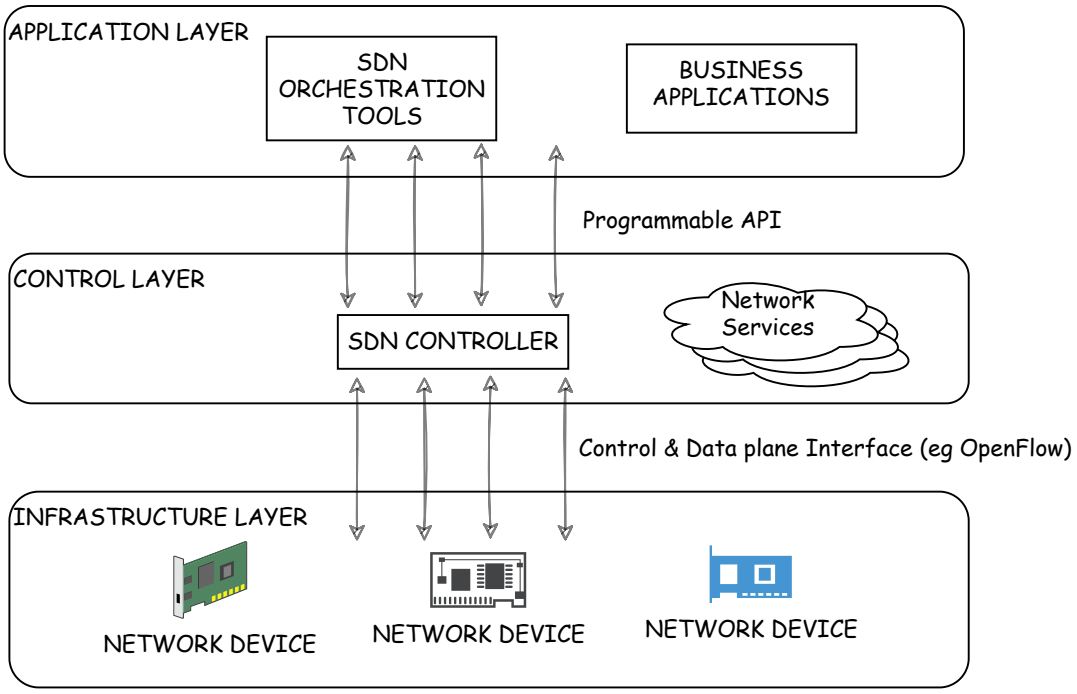
```
apiVersion: template.openshift.io/v1

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
```


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

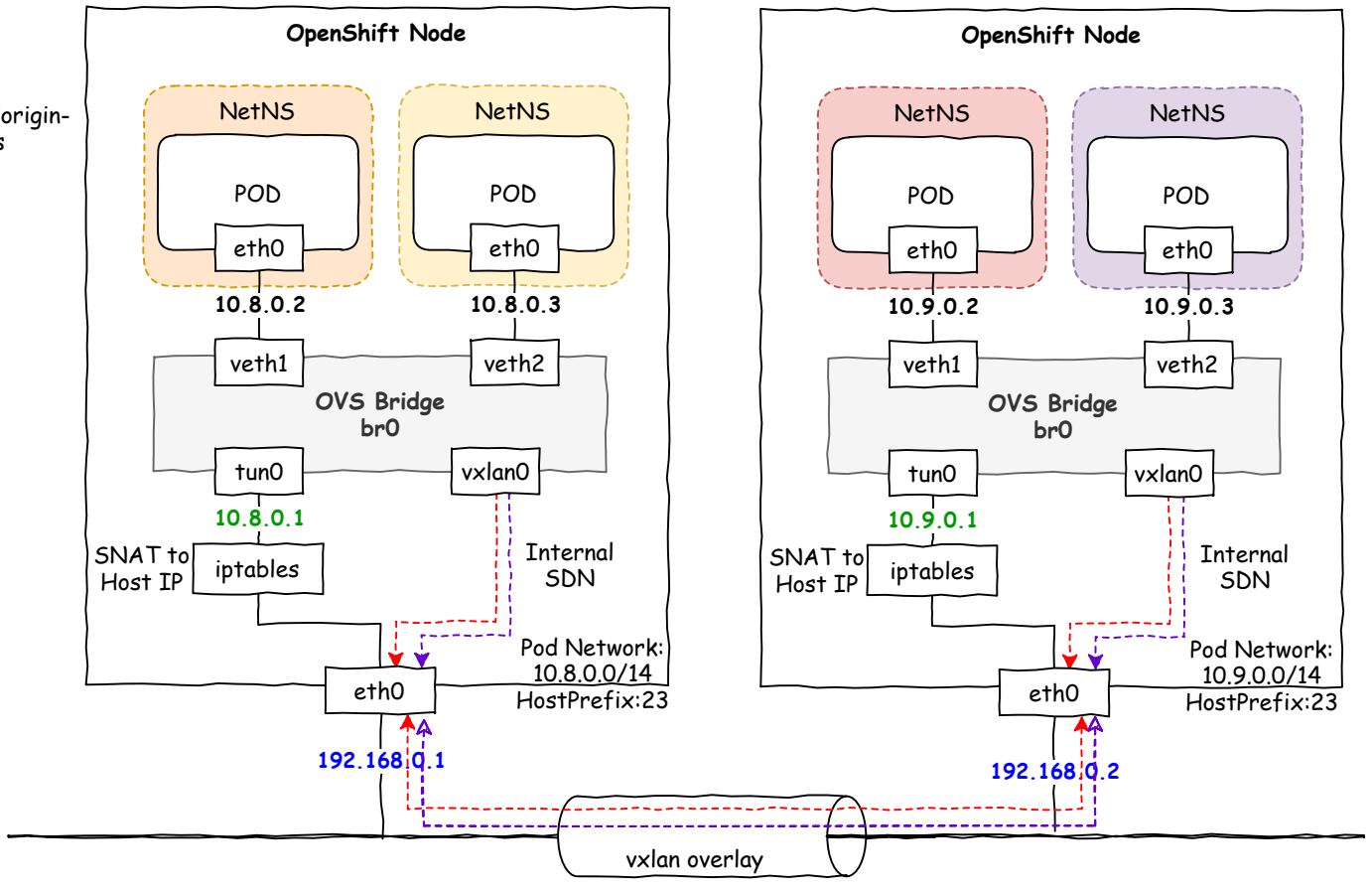

oc label namespace PROJ_A my_func=webapp
oc label namespace PROJ_B my_func=db_server
PROJ_A
Label
my_func=webapp
POD
Label
pname=tst
PROJ_B
POD
Label
app=mydb
Label
my_func=db_server
oc explain NetworkPolicy.spec
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
name: mynet_policy
spec: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

# SDN

- Abstraction of network layers
- Decouple network control and forwarding functions




use openshift/origin-tools




**nnnnnnnn . nnnnnnxx . xxxxxxxx . xxxxxxxx**

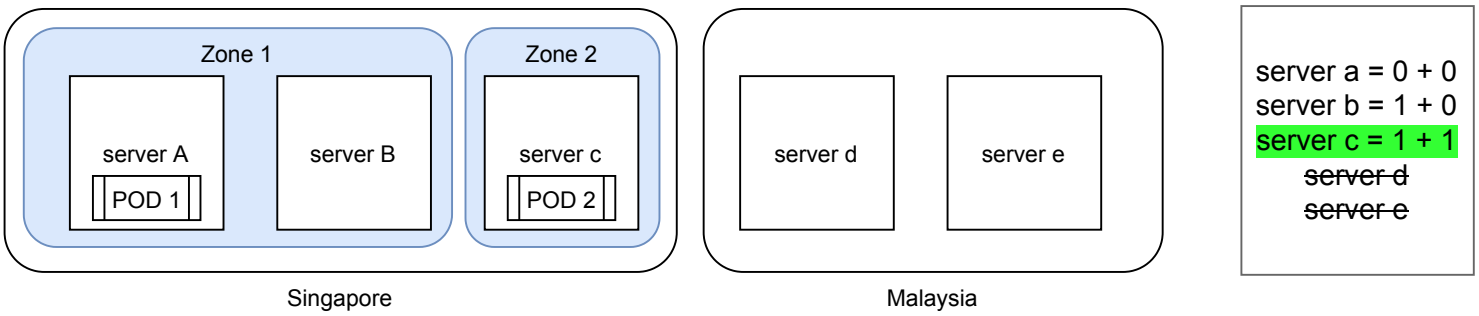
# POD Scheduling

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