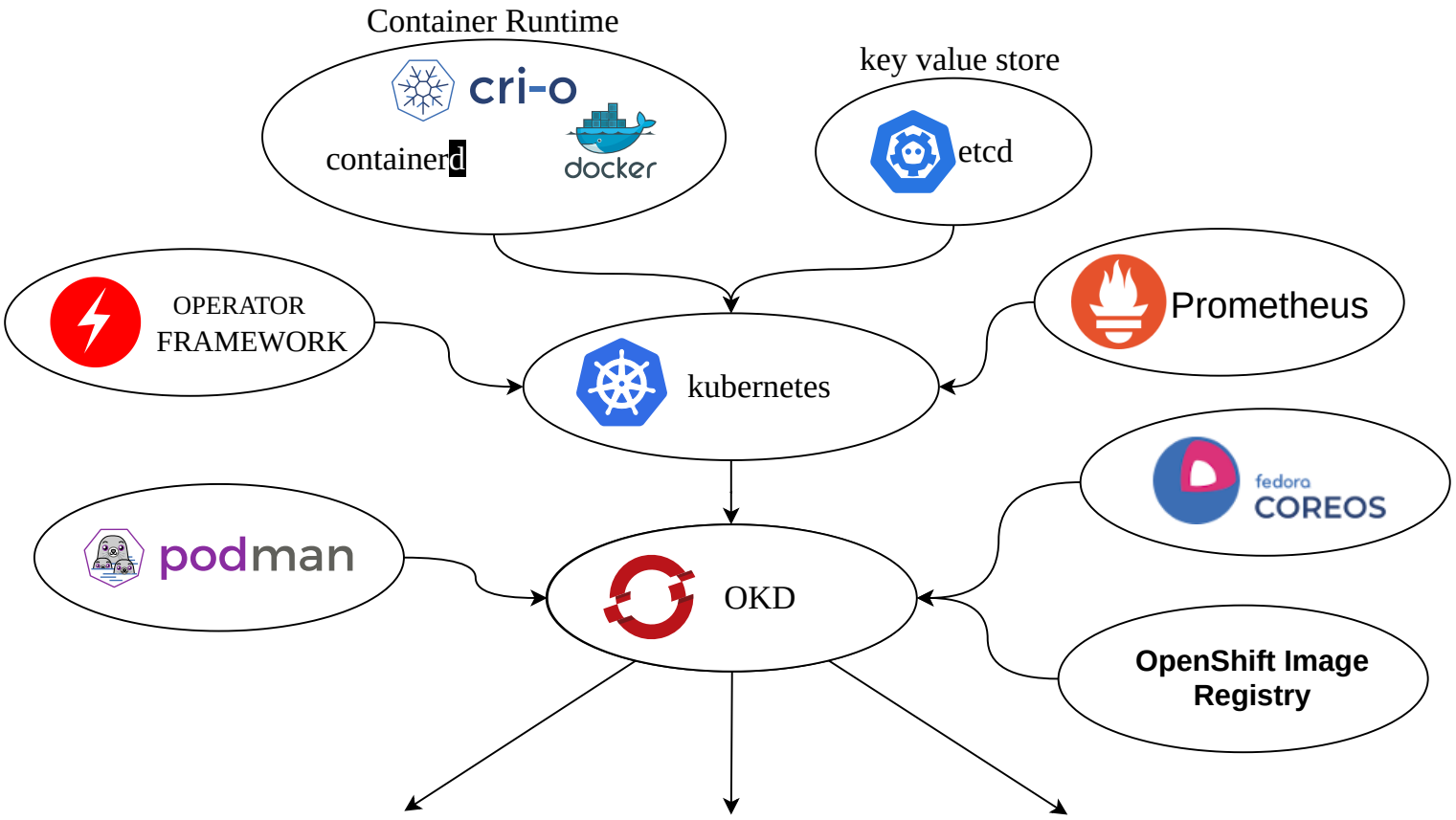


Red Hat OpenShift Container Platform

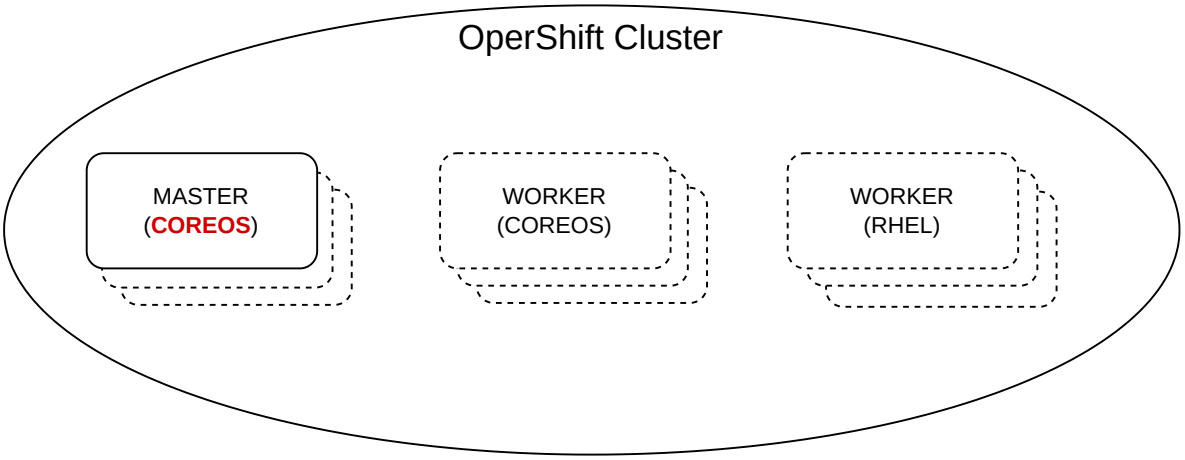
- Public/private DC.
- Bare metal and multiple cloud and virtualization providers.
- Full control by customer.

Red Hat OpenShift Dedicated

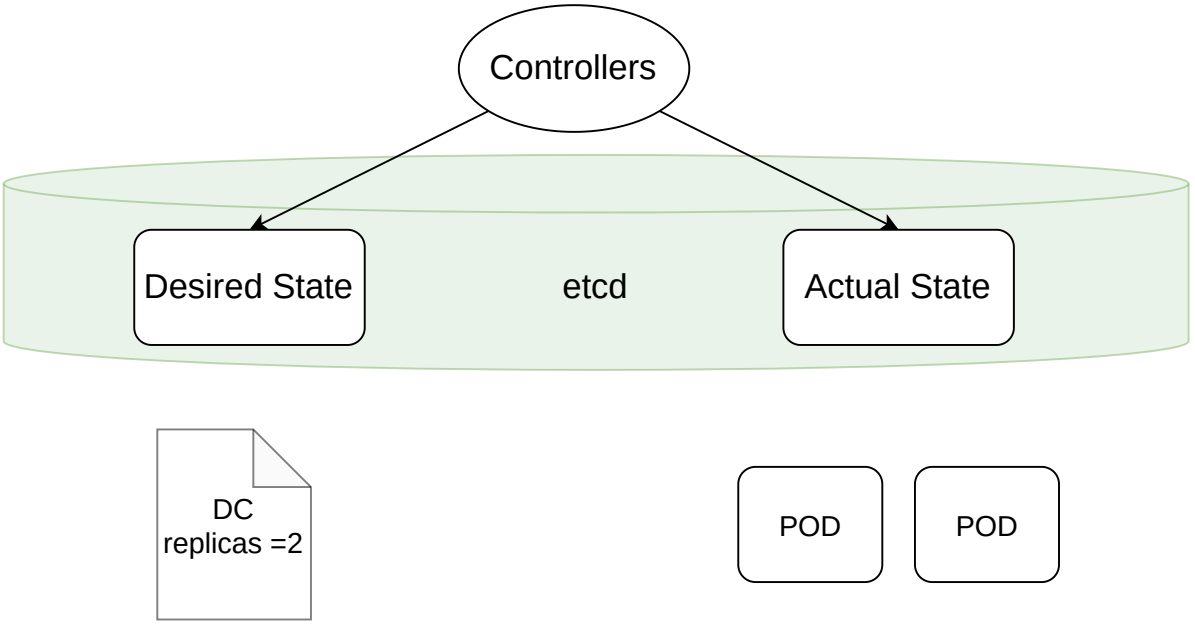
- Managed cluster in public cloud.
- RH manages the cluster.
- Customer manages updates and add-on services.

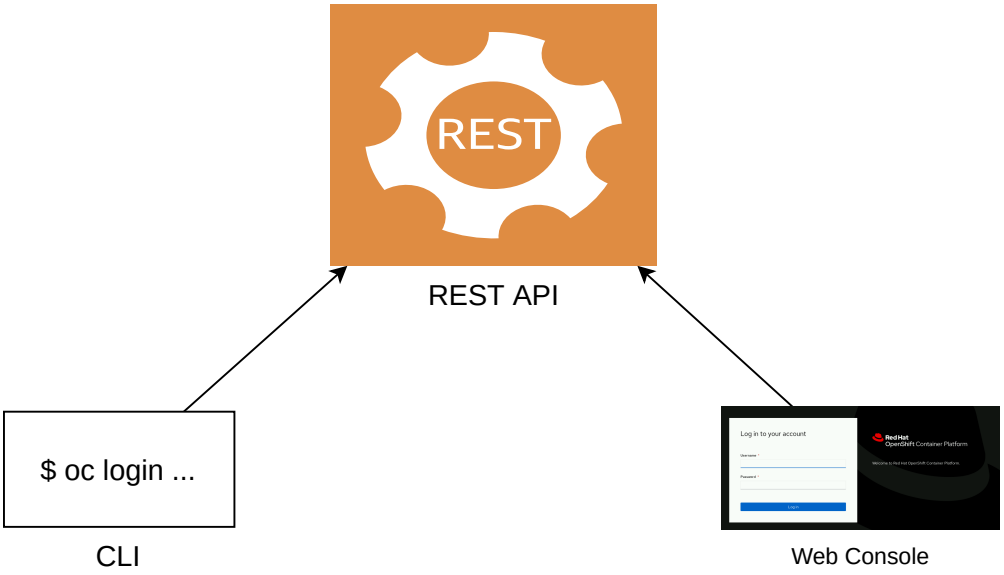
Red Hat OpenShift Online

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

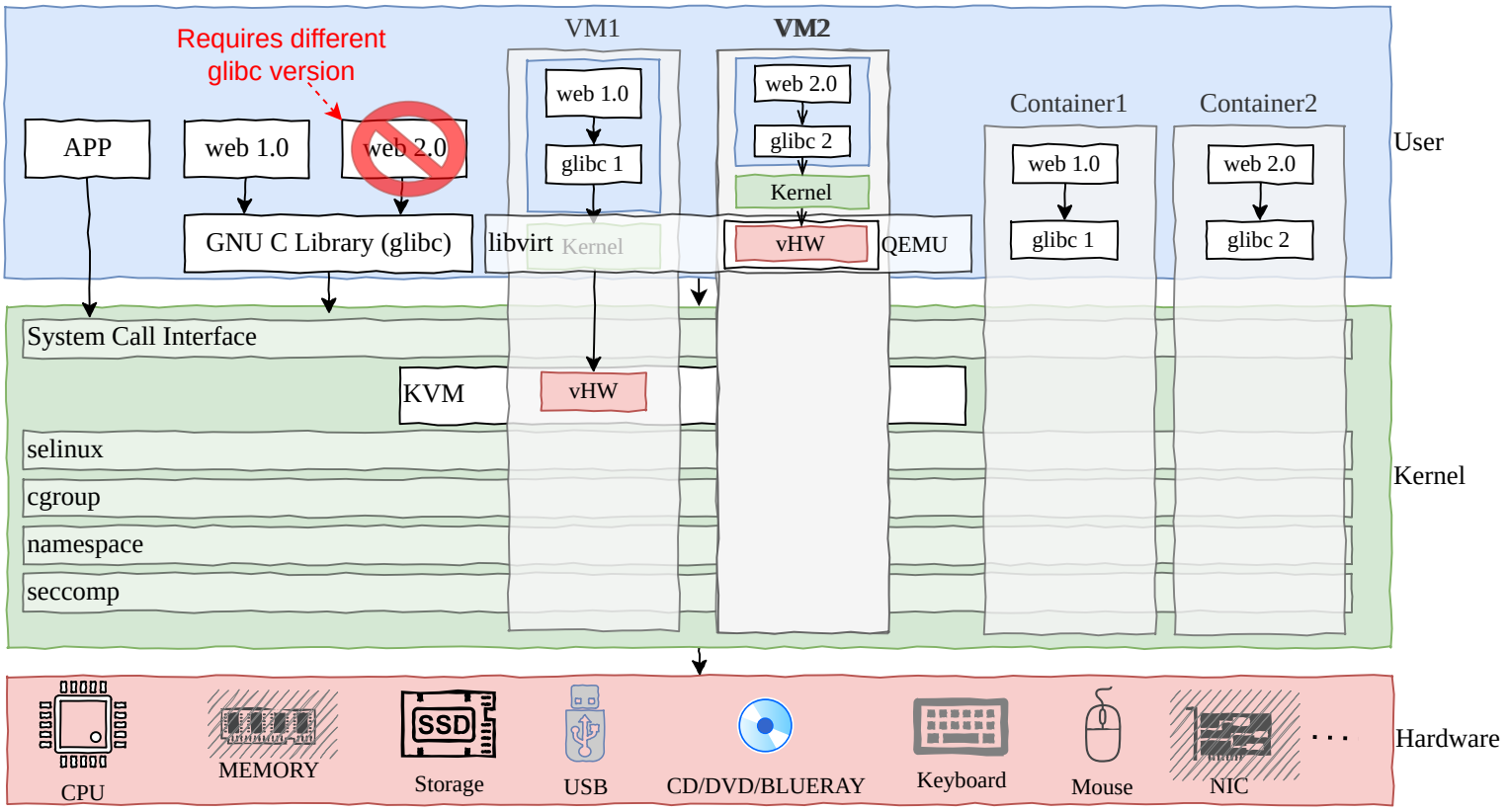


Kubernetes Declarative Architecture

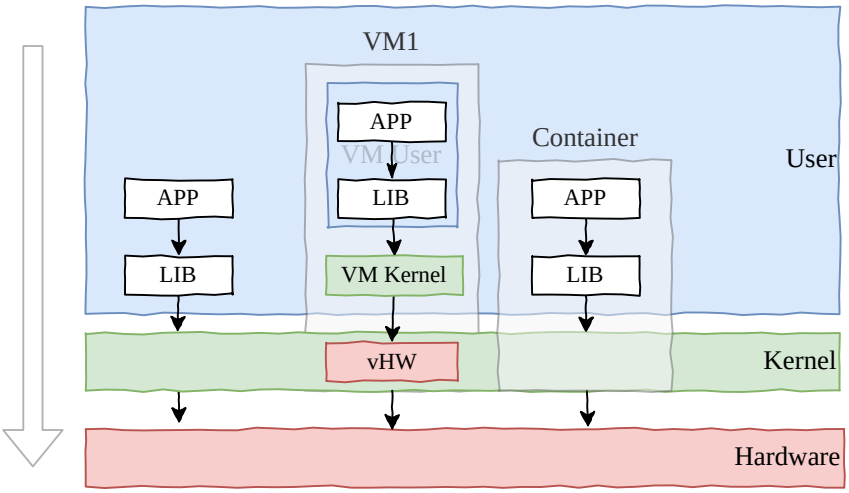




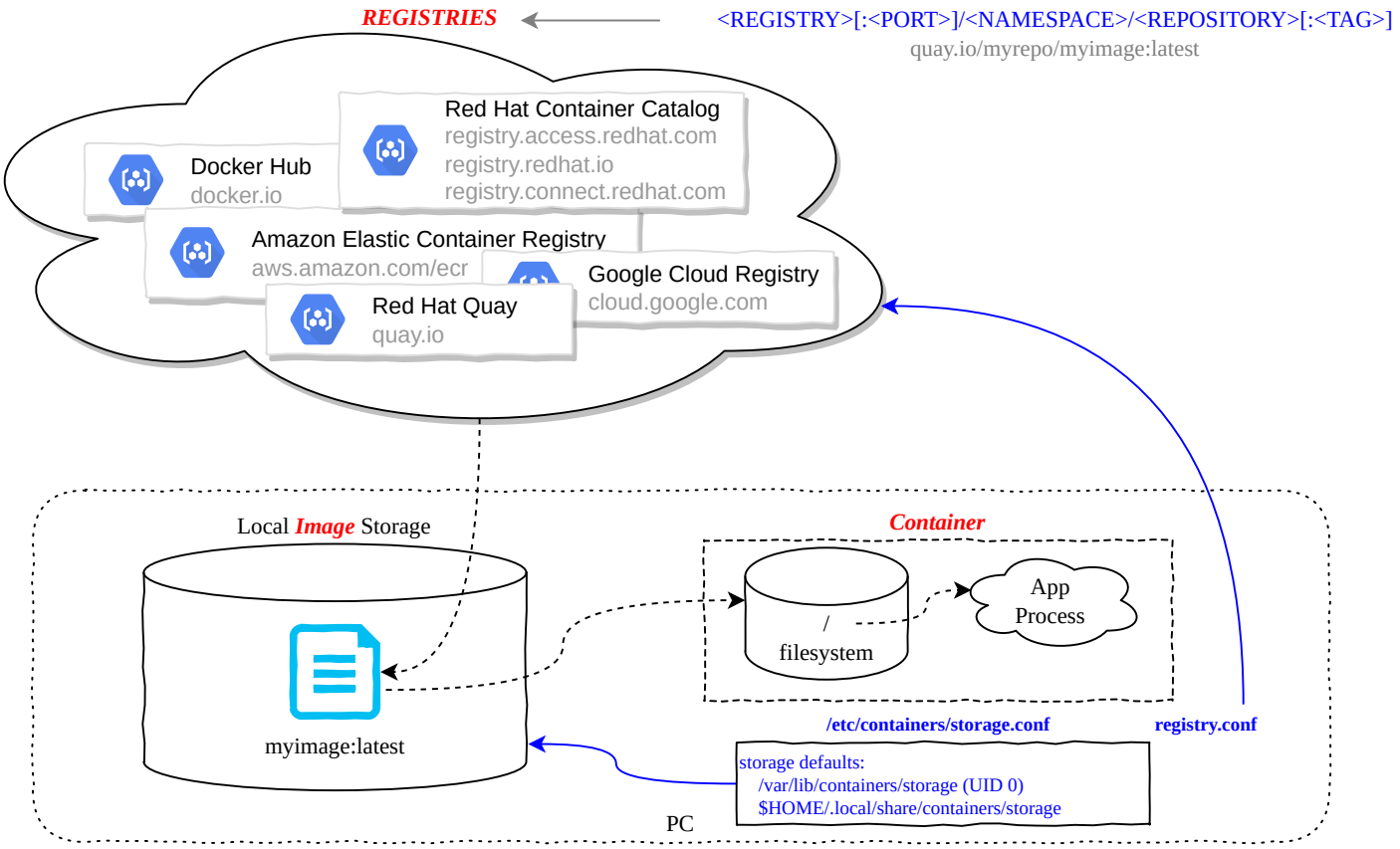
VM vs Container

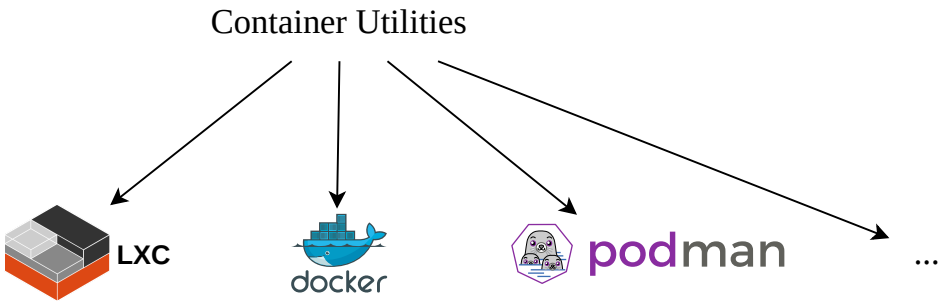


Ref: https://www.redhat.com/en/blog/all-you-need-know-about-kvm-userspace
https://www.packetcoders.io/what-is-the-difference-between-qemu-and-kvm/

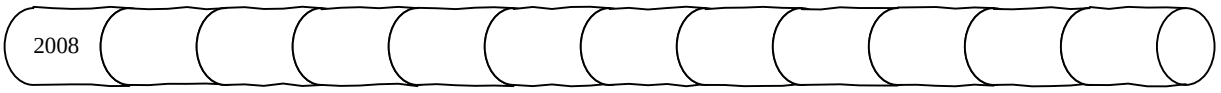


Container Architecture



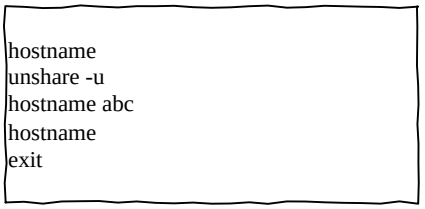


OS Container Vs Application Containers



https://developer.ibm.com/tutorials/multi-architecture-cri-o-container-images-for-red-hat-openshift/

http://www.haifux.org/lectures/299/netLec7.pdf
https://kernelnewbies.org/Linux_2_X_XX



Heading

man 7 namespaces

- mount (2.4.19) - 3/8/2002 - CAP_SYS_ADMIN

- pid (2.6.24) - 24/1/2008

- net (2.6.29) - 23/3/2009

- ipc (2.6.19) 29/11/2006

- uts (2.6.19) 29/11/2006

- user (3.8) 18/2/2013 no cap

- cgroup (4.6) 15/5/2016

- time - 3/2020

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.

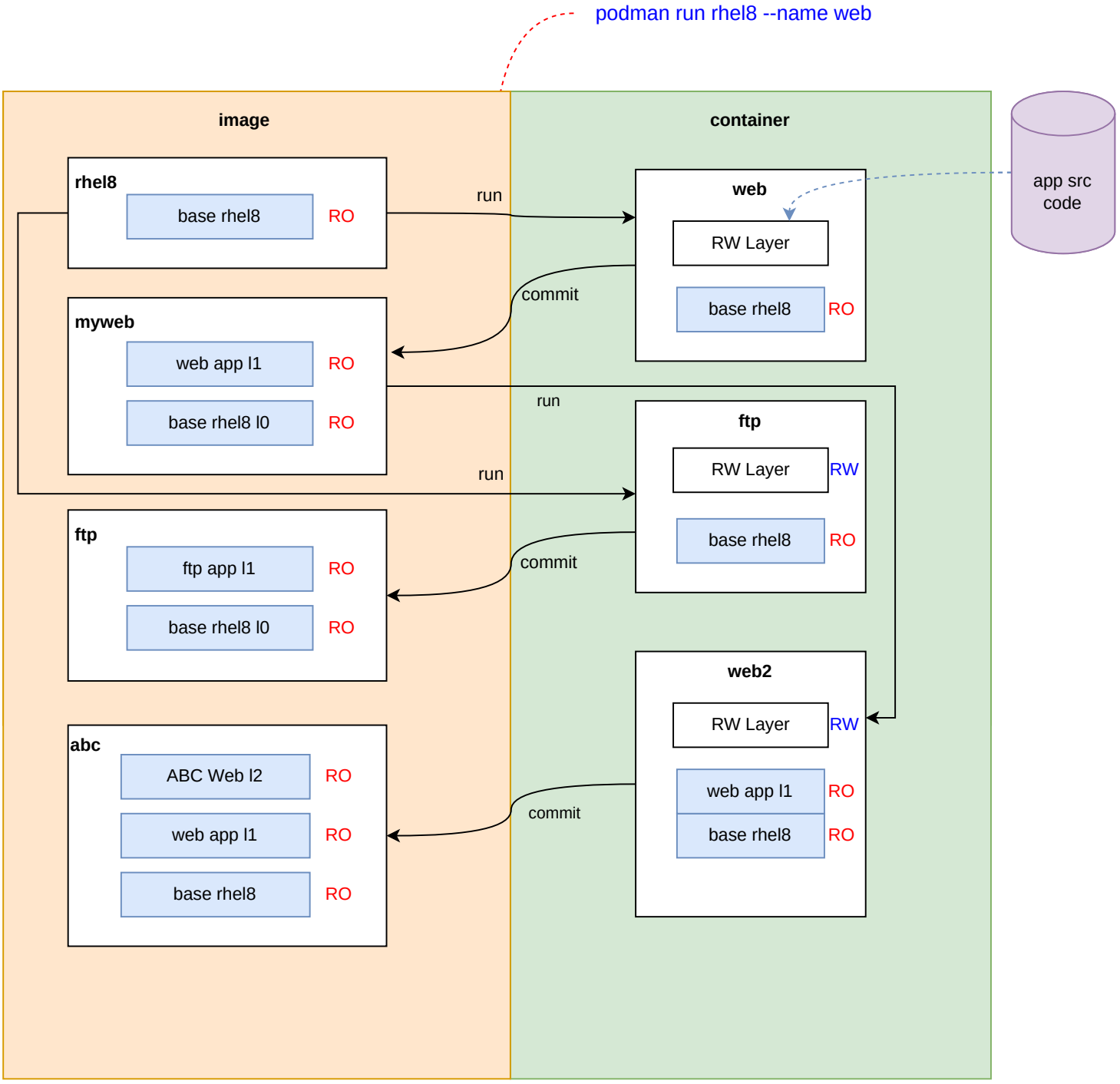
For more info:

podman --help OR **man podman**

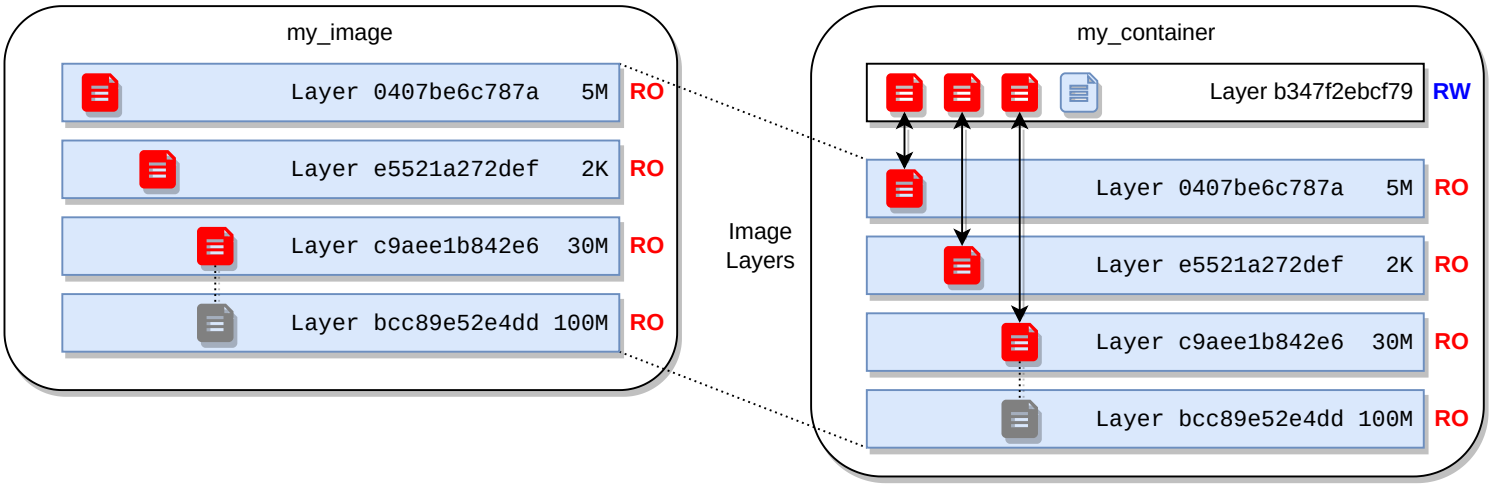
Each sub command has it's own man page. i.e man podman-run, man podman-images, etc.

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

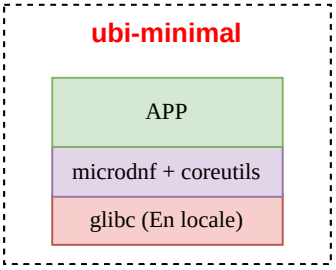


UnionFS - A Stackable Unification File System



BASE IMAGE TYPES

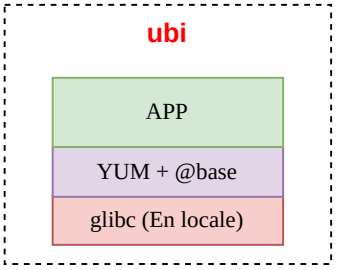
MINIMAL



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)

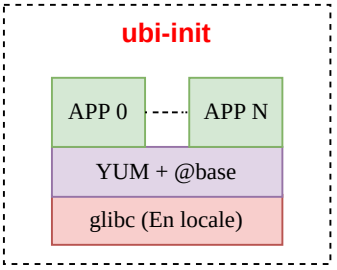
PLATFORM



For any apps that runs on RHEL

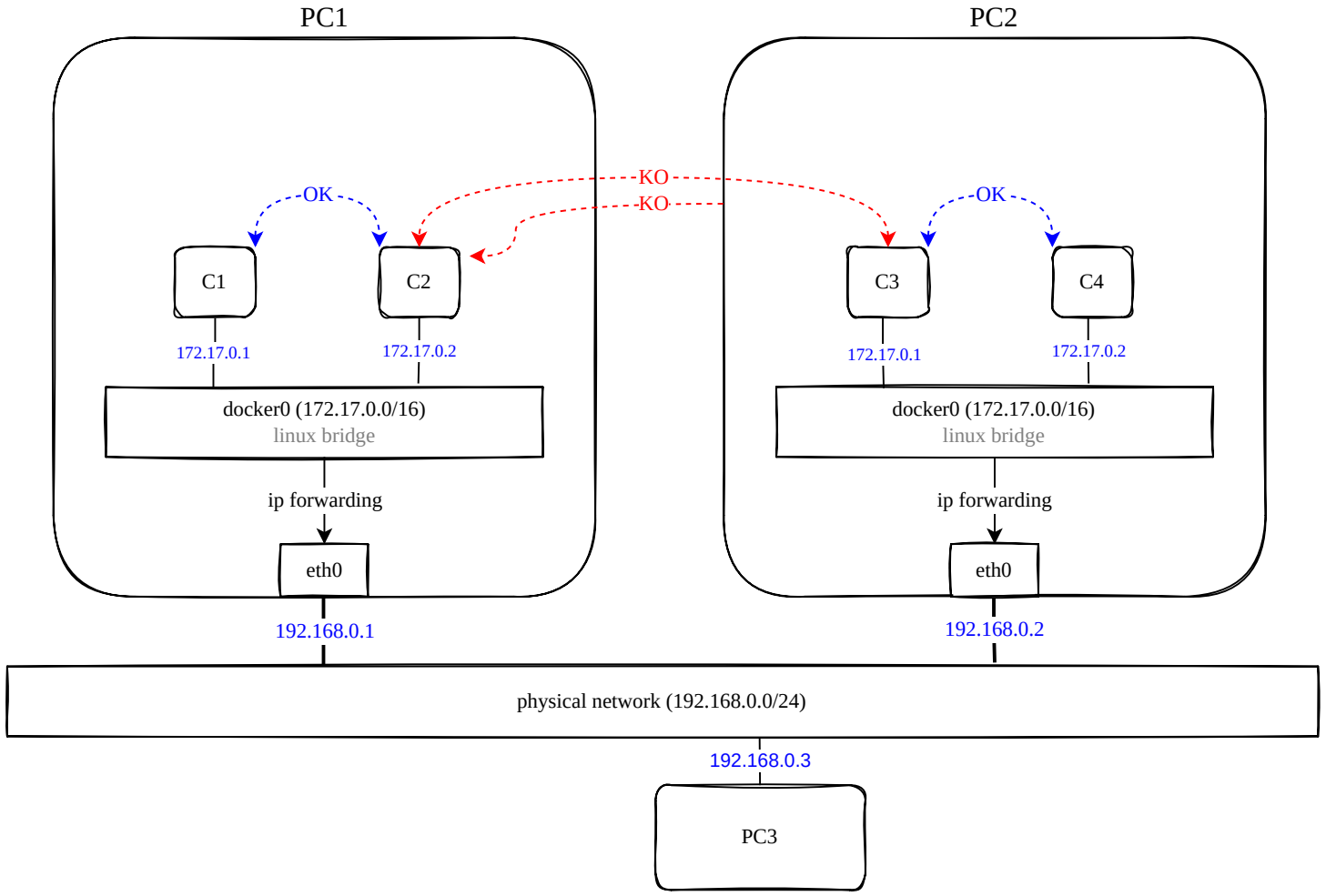
- Unified, OpenSSL crypto stack
- Full YUM stack
- Includes useful basic OS tools (tar, gzip, vi, etc)

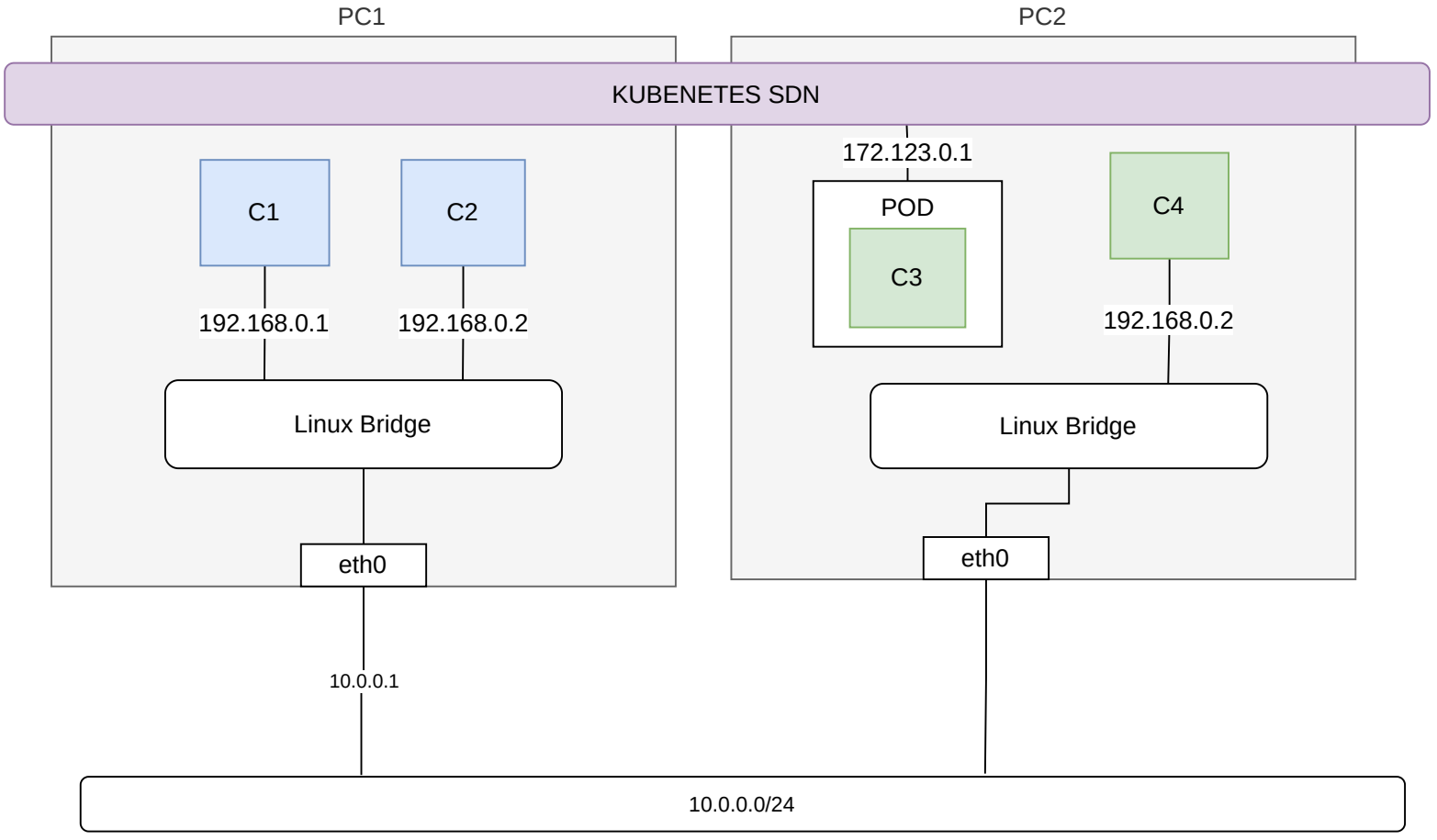
MULTI-SERVICE



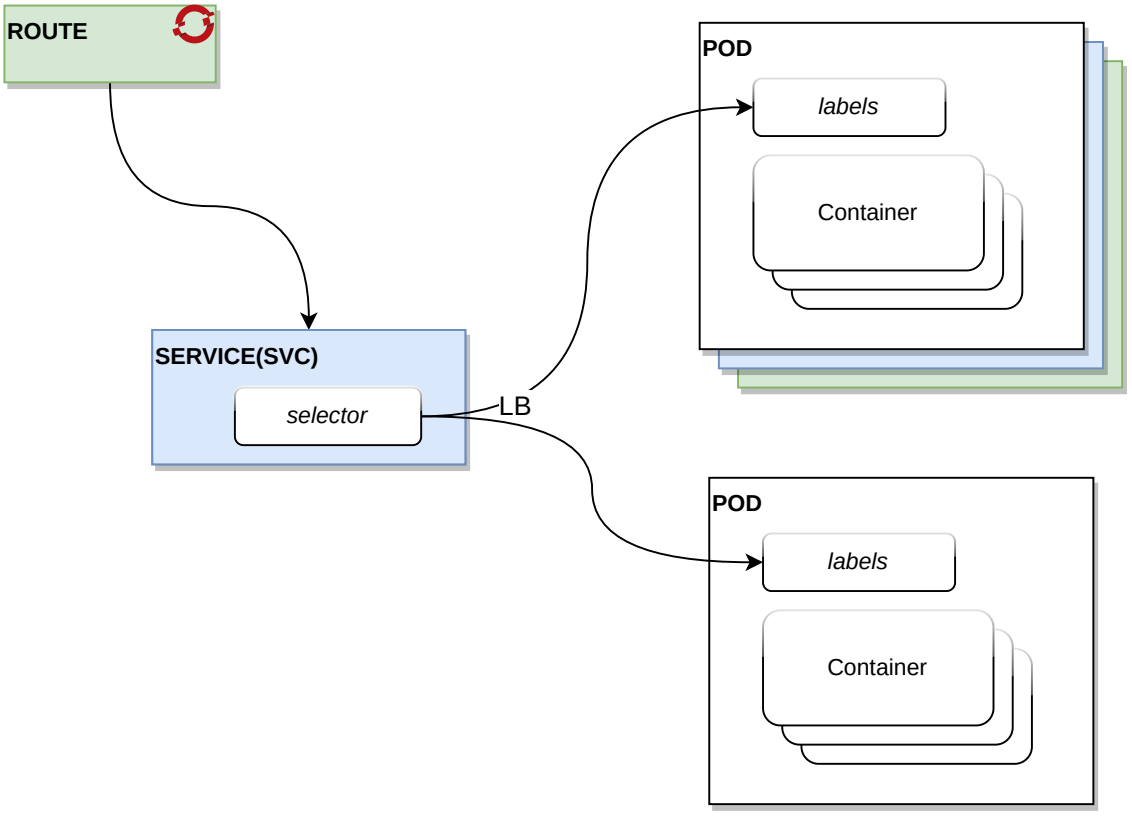
Eases running multi-service in single container

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


PC1
PC2
OK
KO
KO
OK
C1
C2
C3
C4
172.17.0.1
172.17.0.2
172.17.0.1
172.17.0.2
docker0 (172.17.0.0/16)
linux bridge
docker0 (172.17.0.0/16)
linux bridge
ip forwarding
ip forwarding
eth0
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



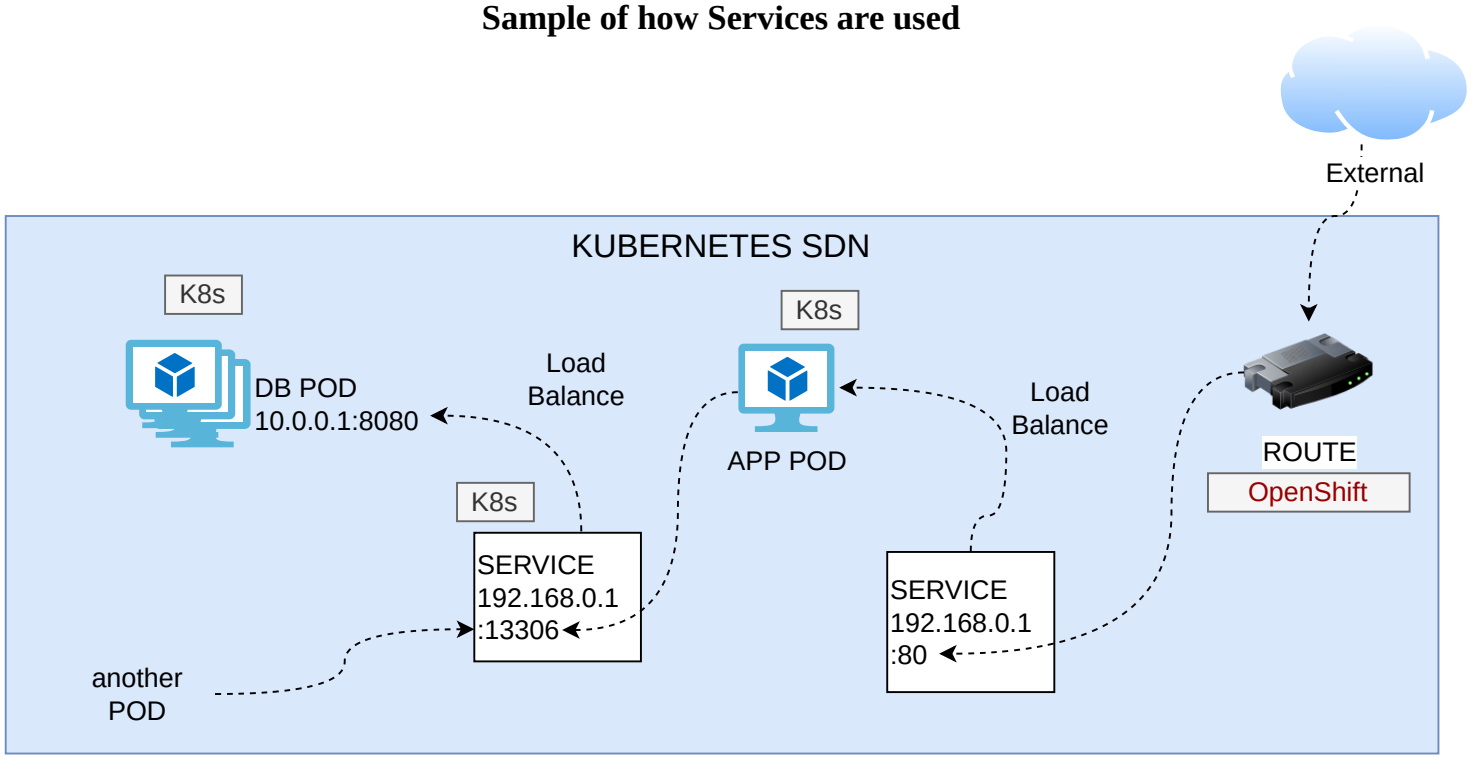
POD
A pod contains one or more containers.

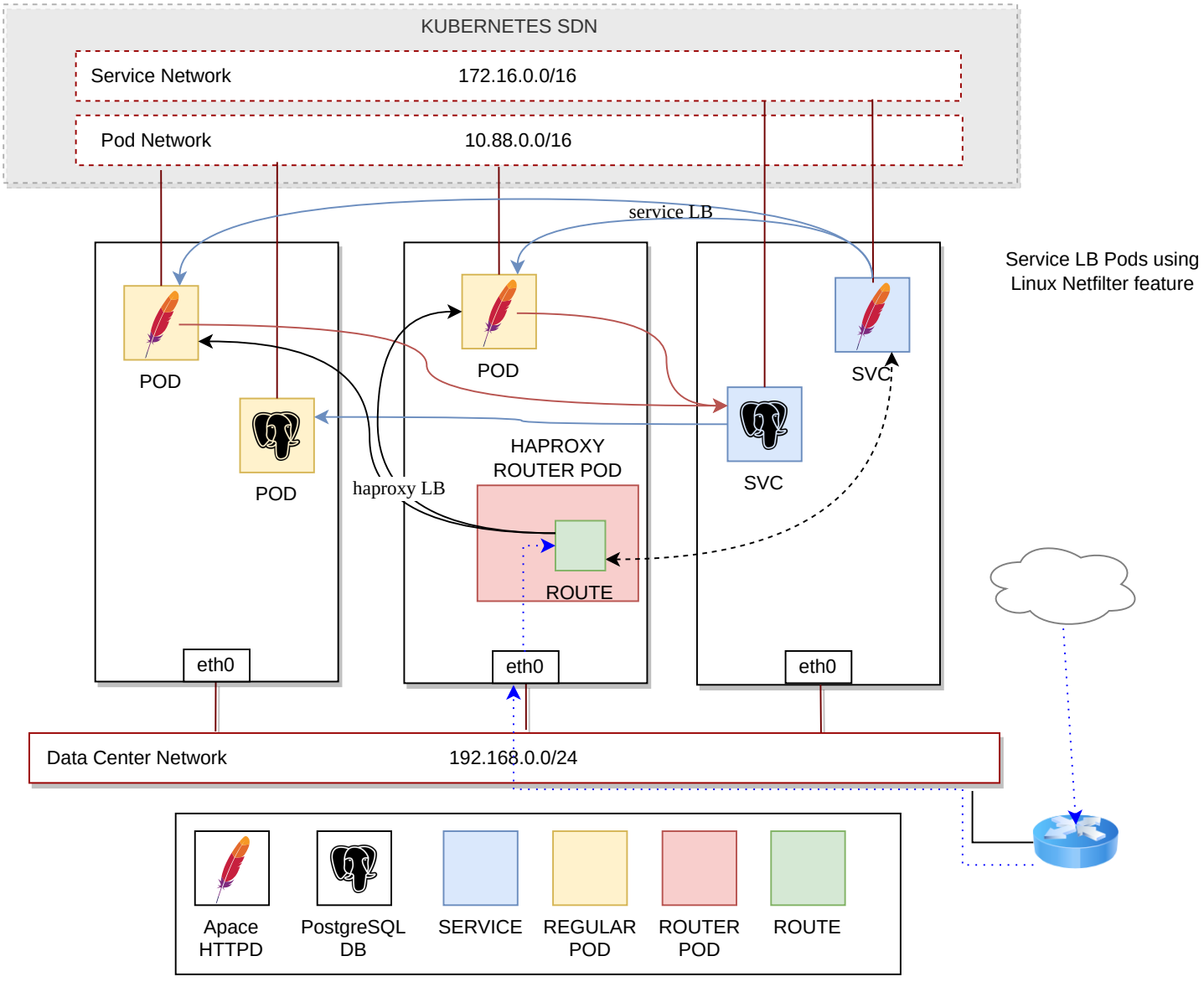
SERVICE
A service references the pod(s) by using the label selector.
The service load balances the connections between all the pods.

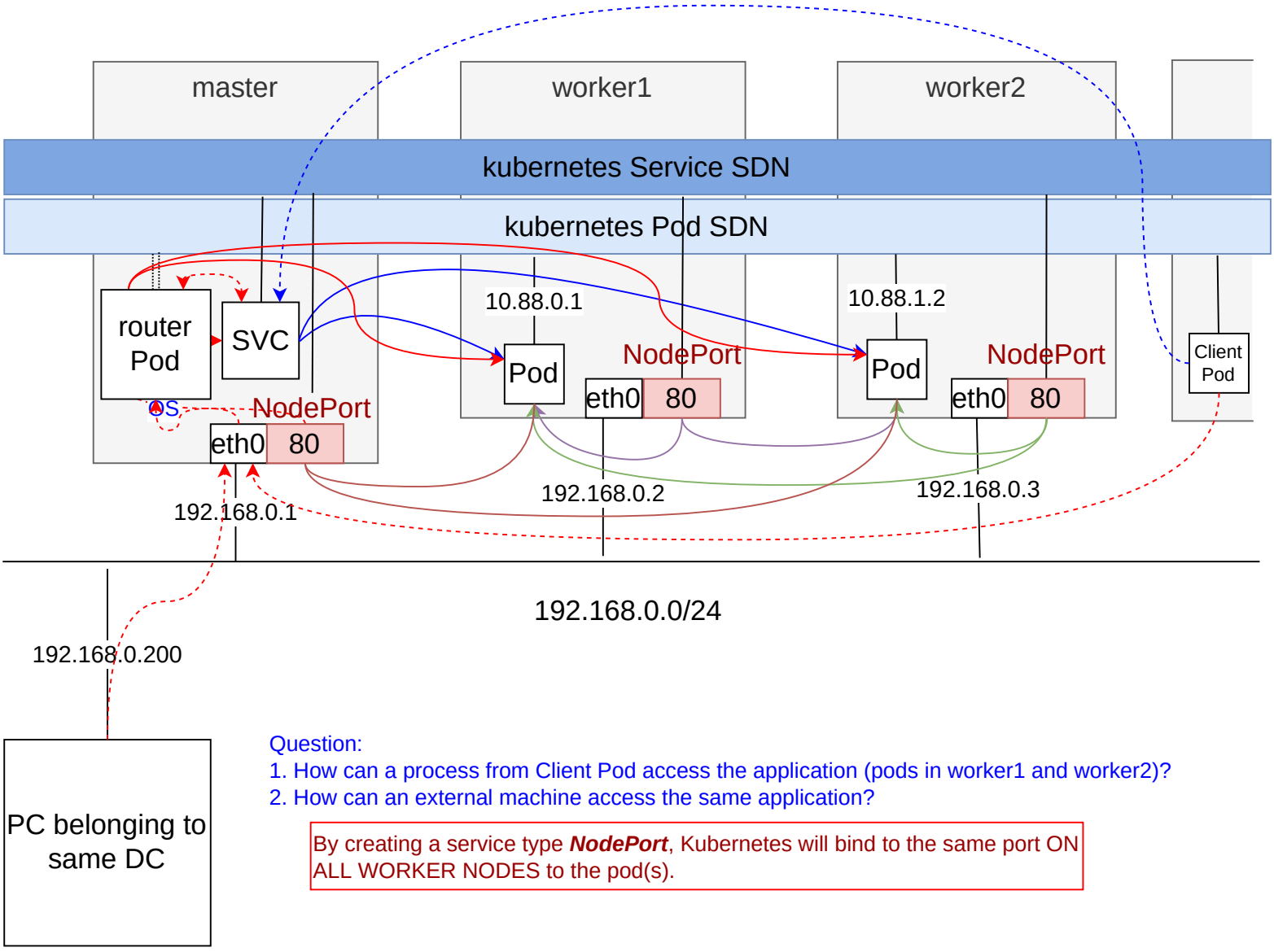
ROUTE
A route exposes the service to the external world.

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

Sample of how Services are used





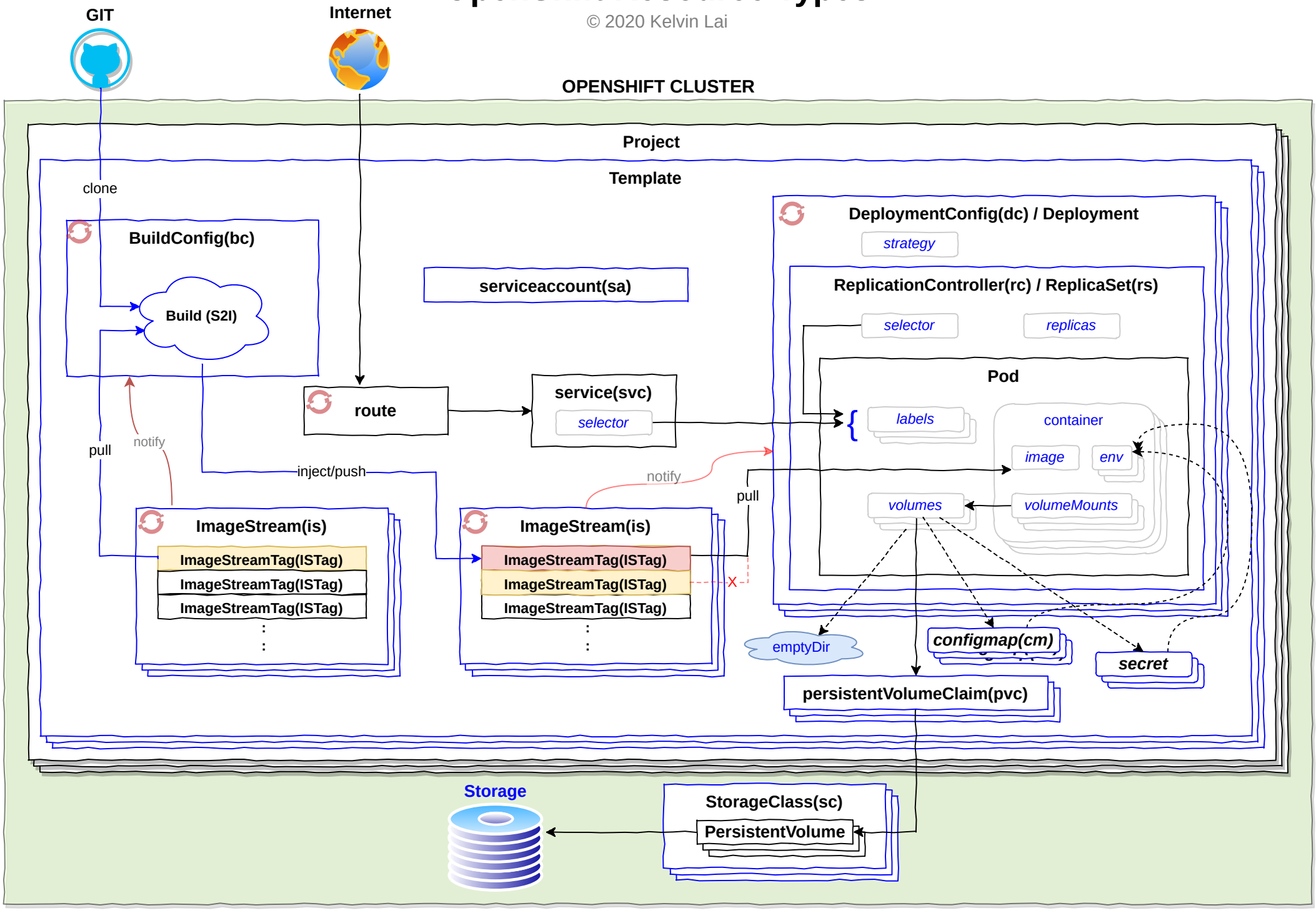


Question:

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?

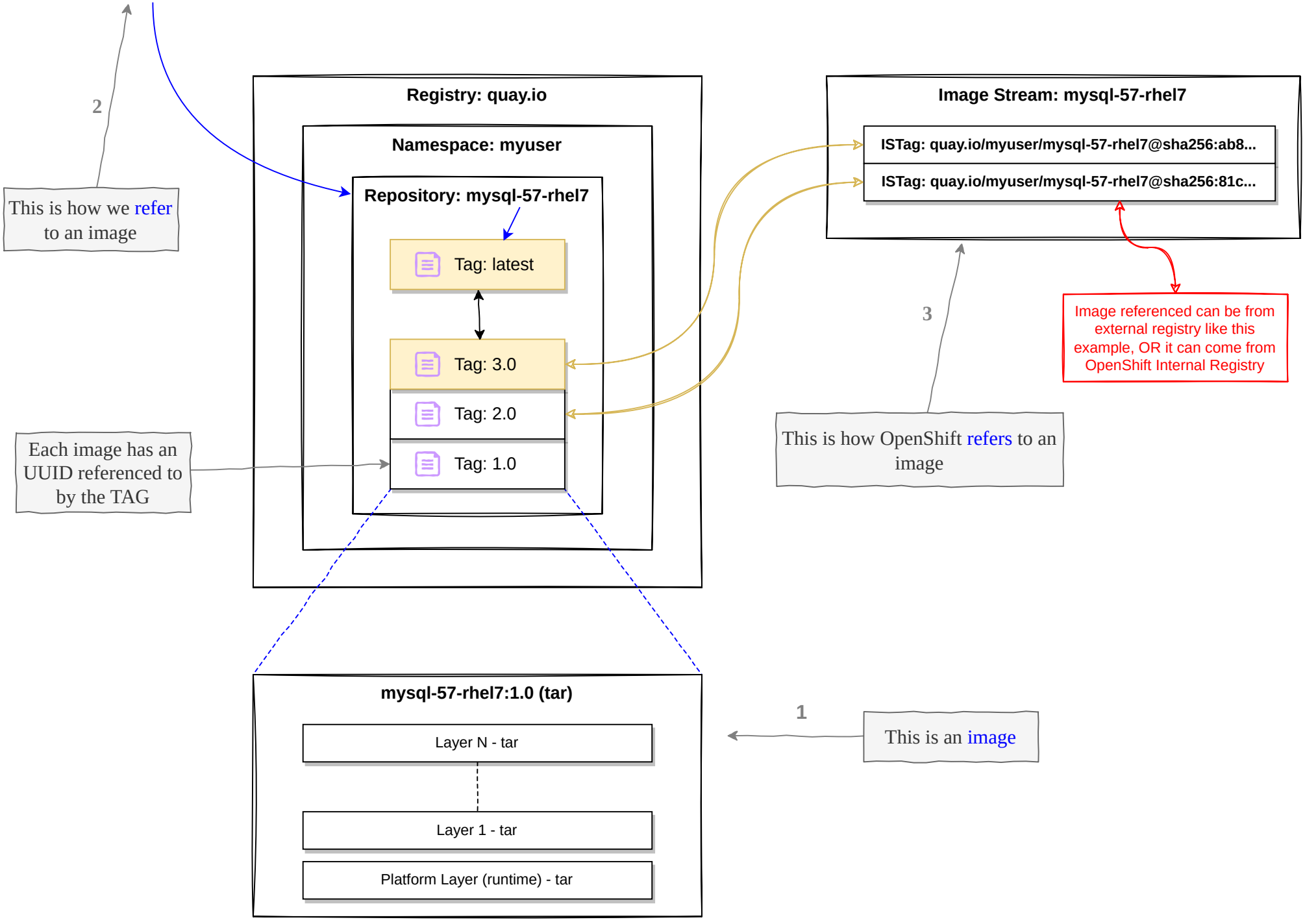
By creating a service type ***NodePort***, Kubernetes will bind to the same port ON ALL WORKER NODES to the pod(s).

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>
```

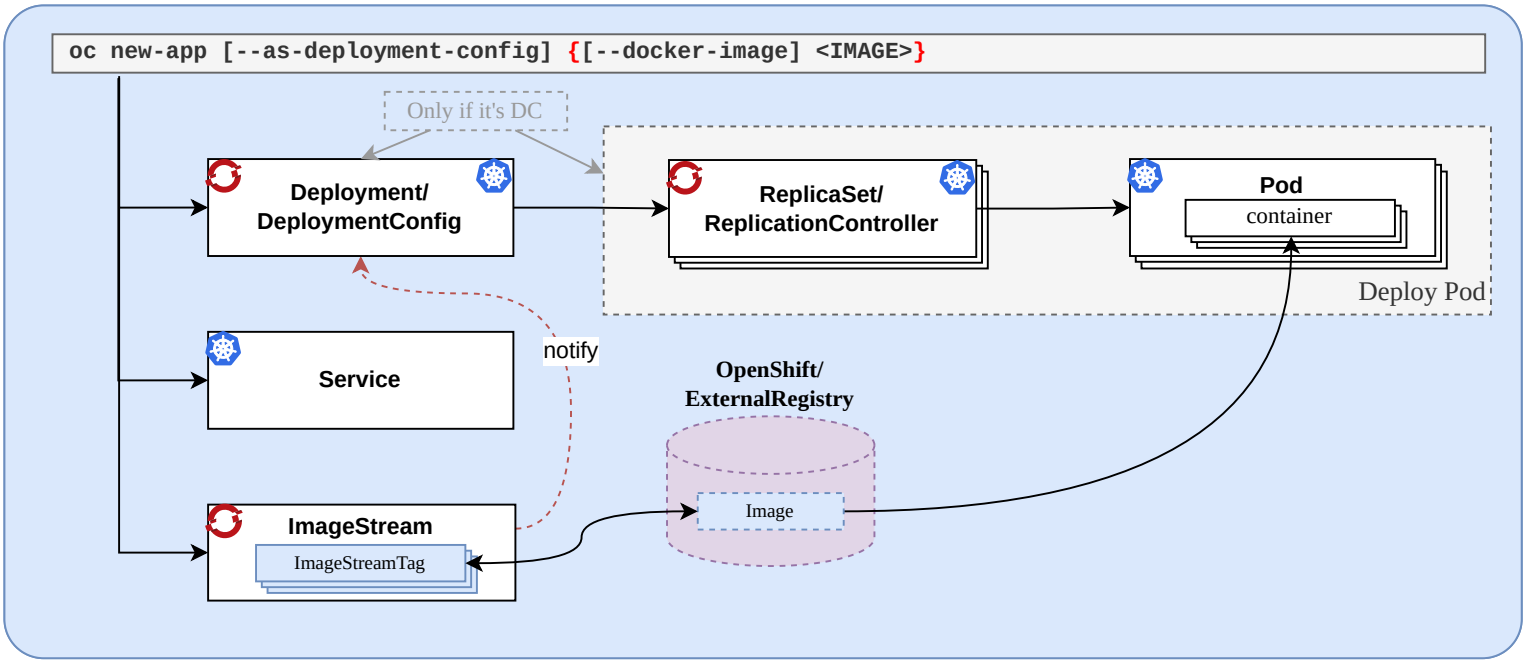
3. Using yaml/json file

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

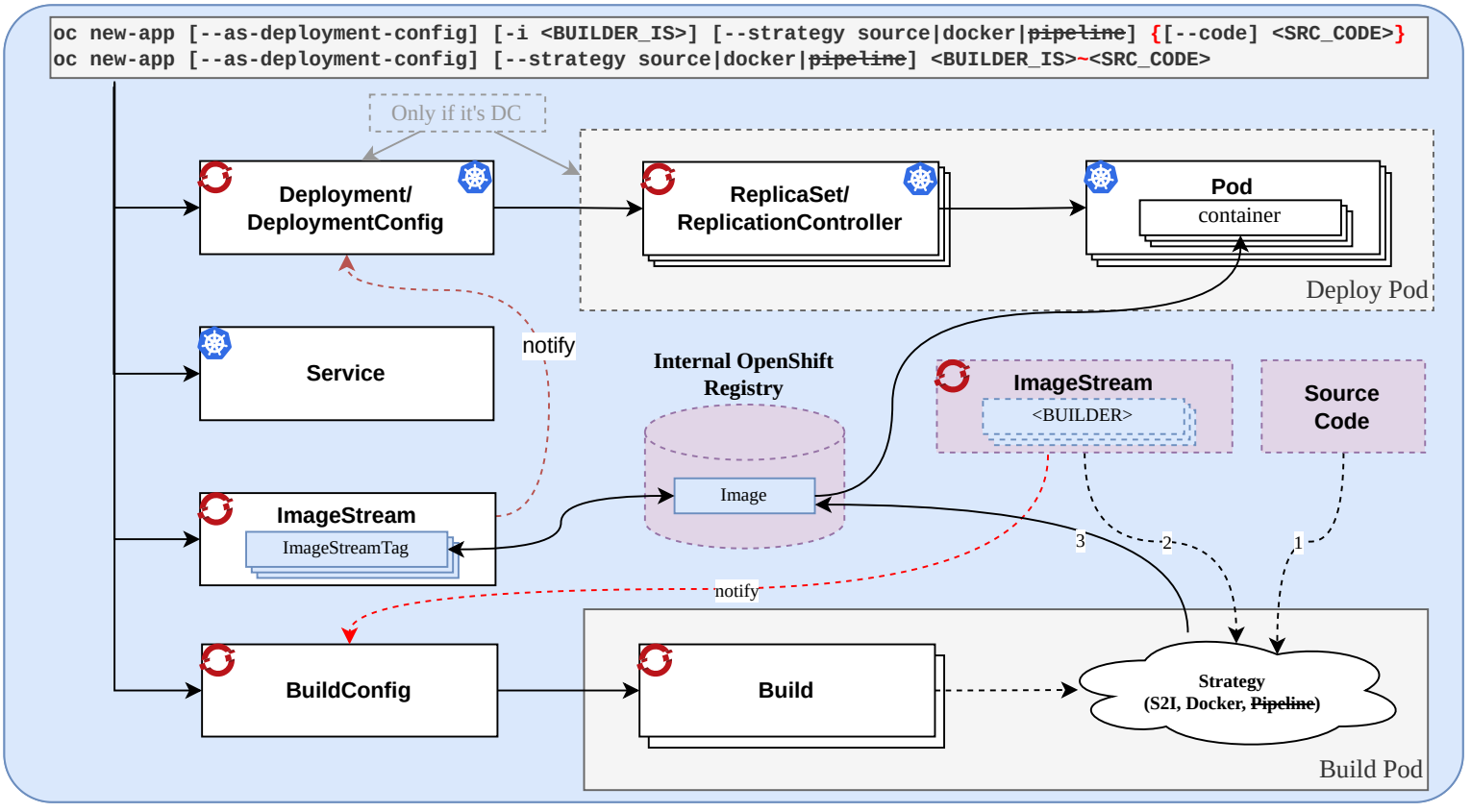
4. Using template

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

1. Use Existing Image



2. Managed Life Cycle



```
oc new-app -i 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.

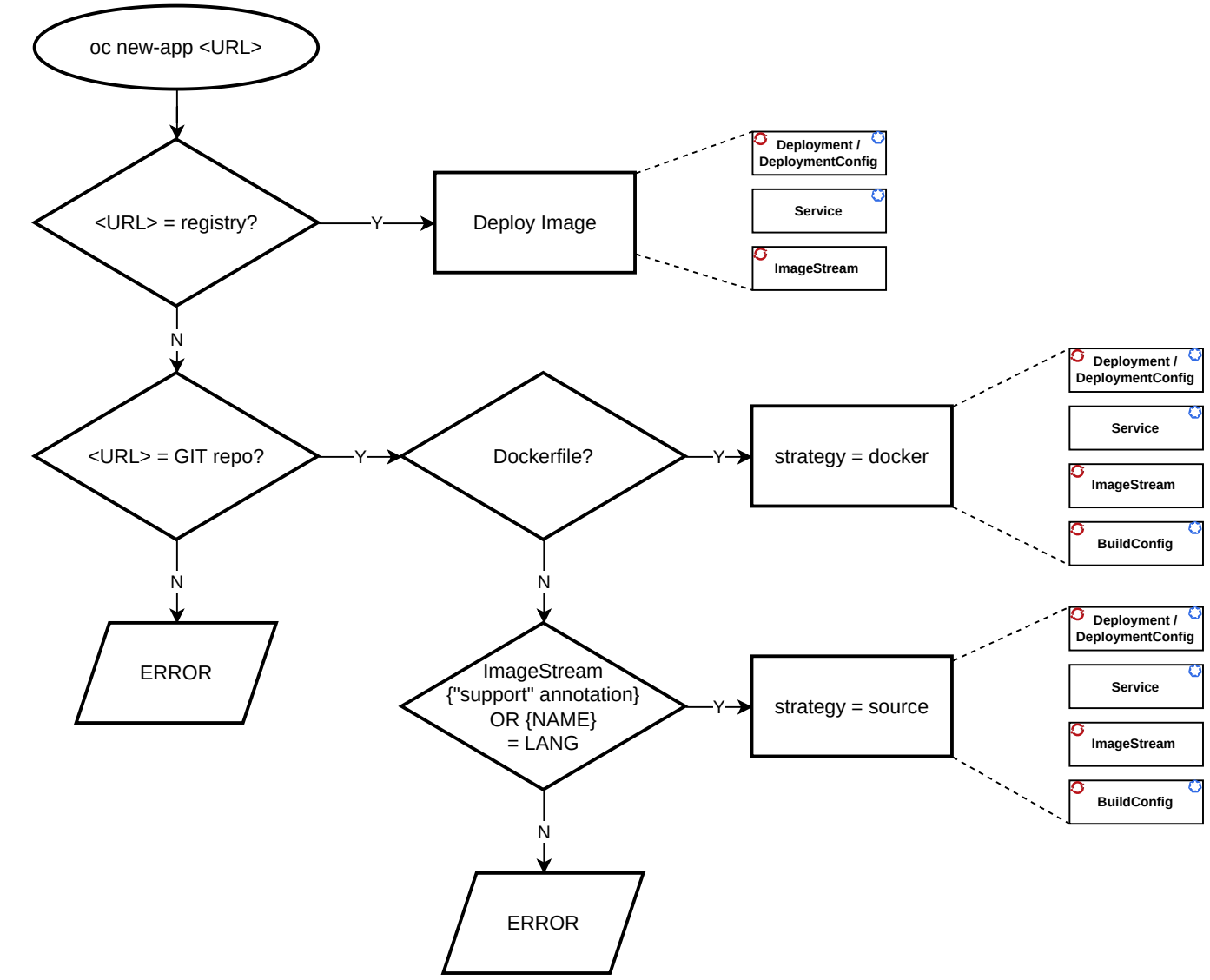
oc expose <DC/DEPLOYMENT/RC/RS/POD> <RESOURCE_NAME>

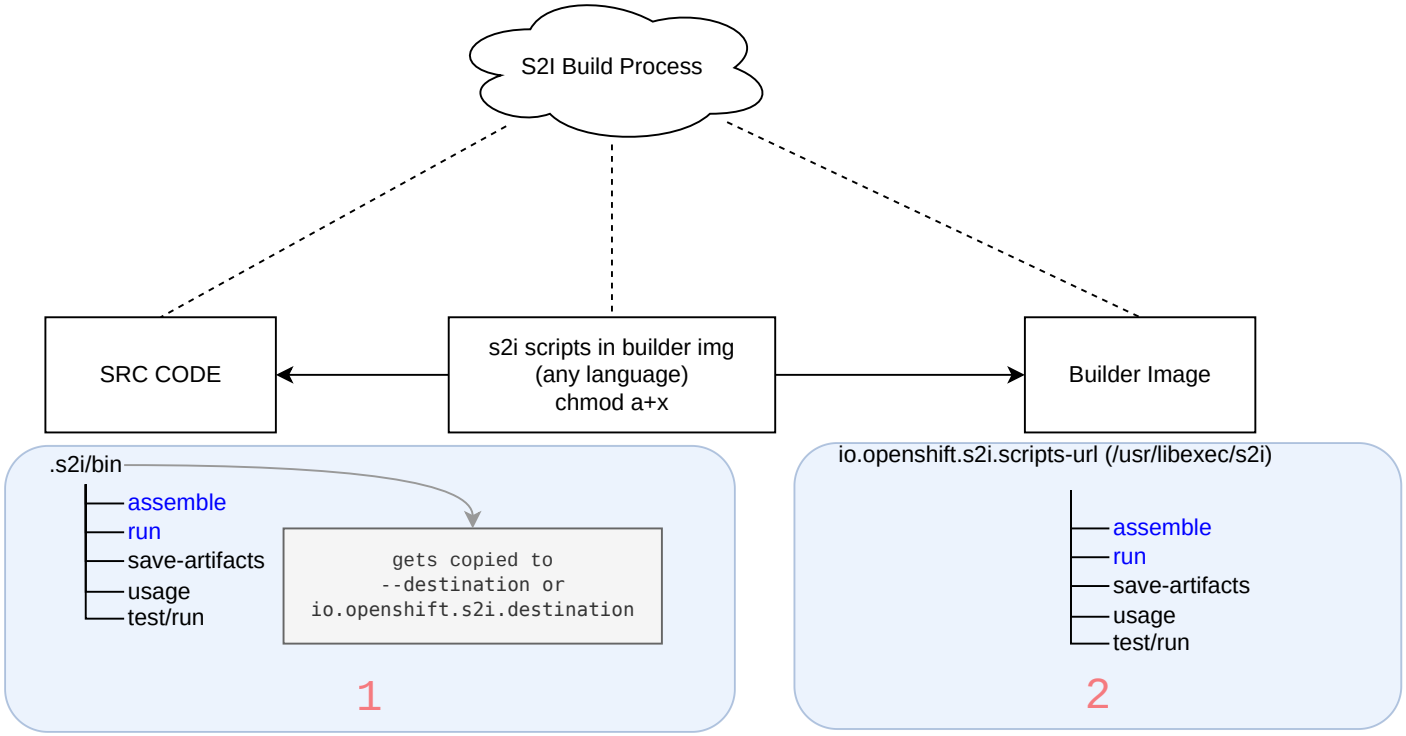
DNS NAME = <SVC>.<PROJ>[.svc.cluster.local]
ENVIRONMENT VARIABLE IN POD = <SVC>_SERVICE_HOST

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

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

oc new-app flowchart





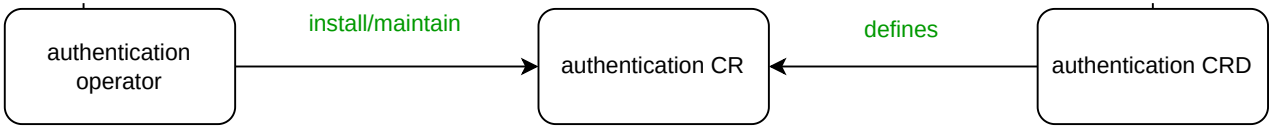
Template

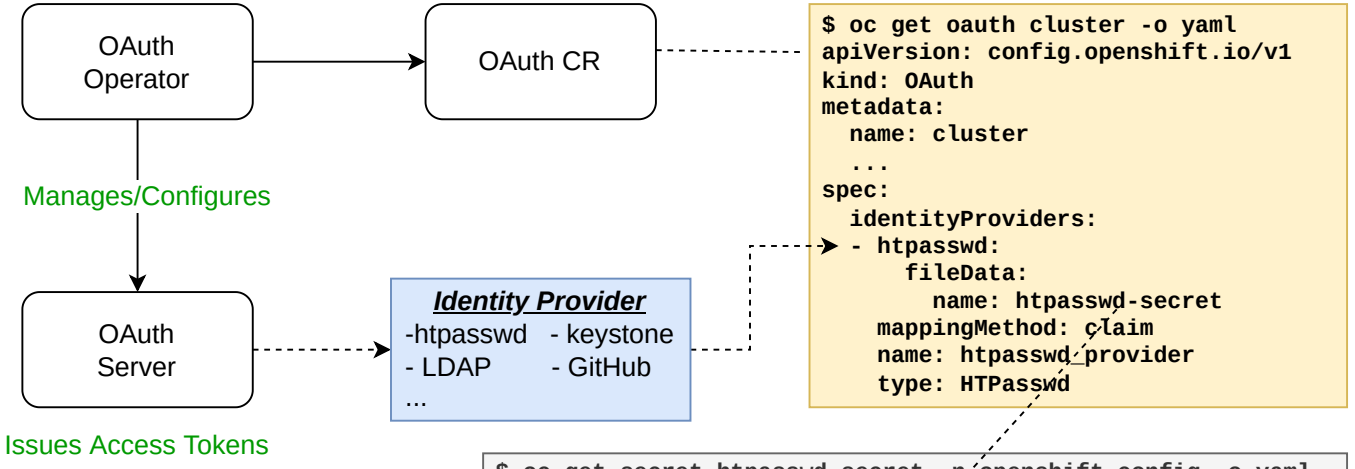
```
apiVersion: template.openshift.io/v1

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
```

```
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...'
      }
    }
  }
}
```

```
$ oc get clusteroperator authentication
NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
authentication   4.5.4     True        False         False      40d
```

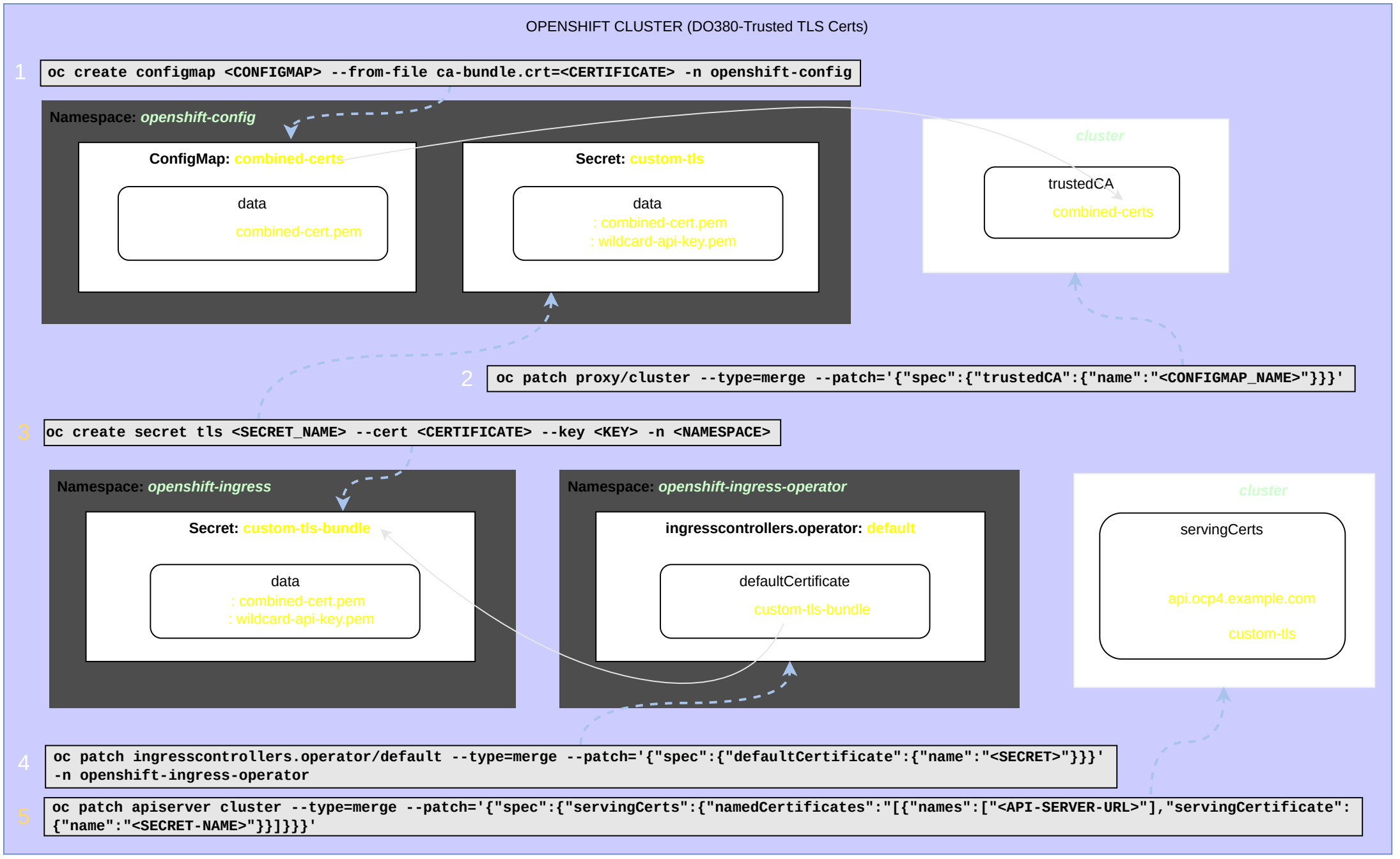
```
$ oc get crd authentications.operator.openshift.io
```

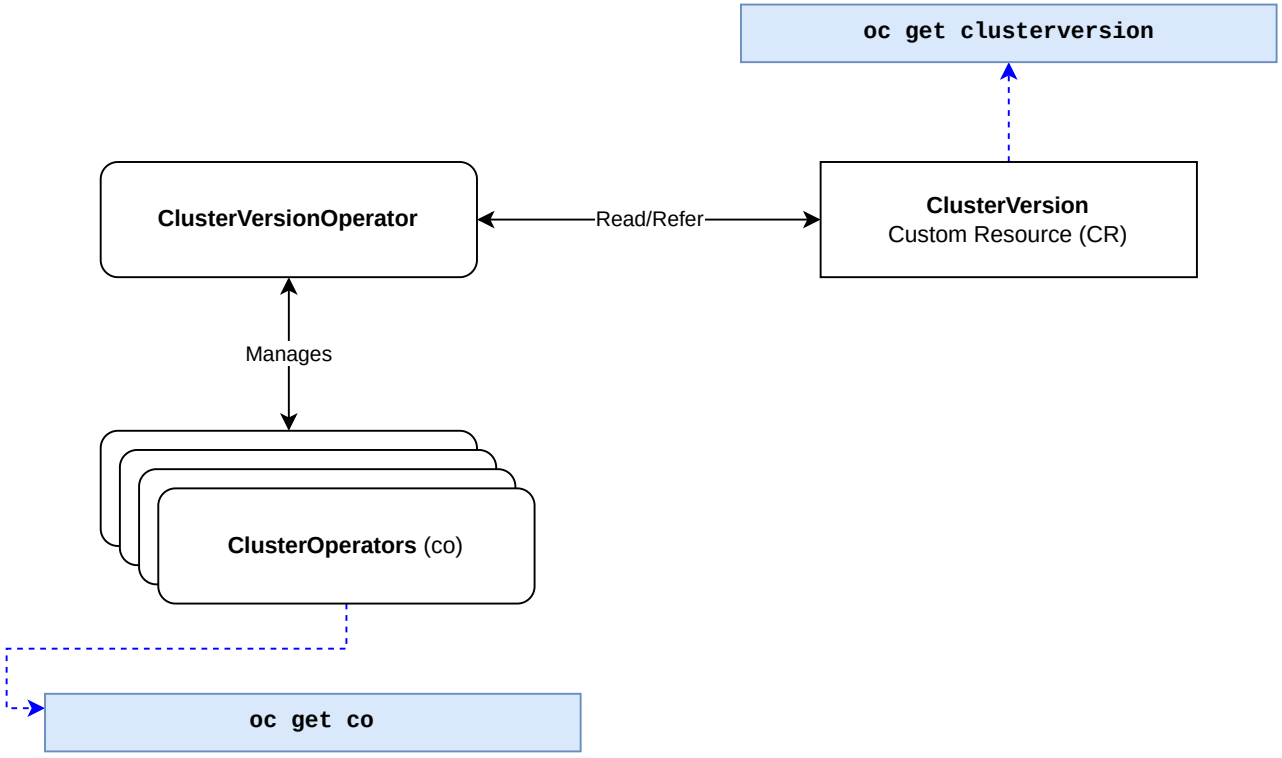




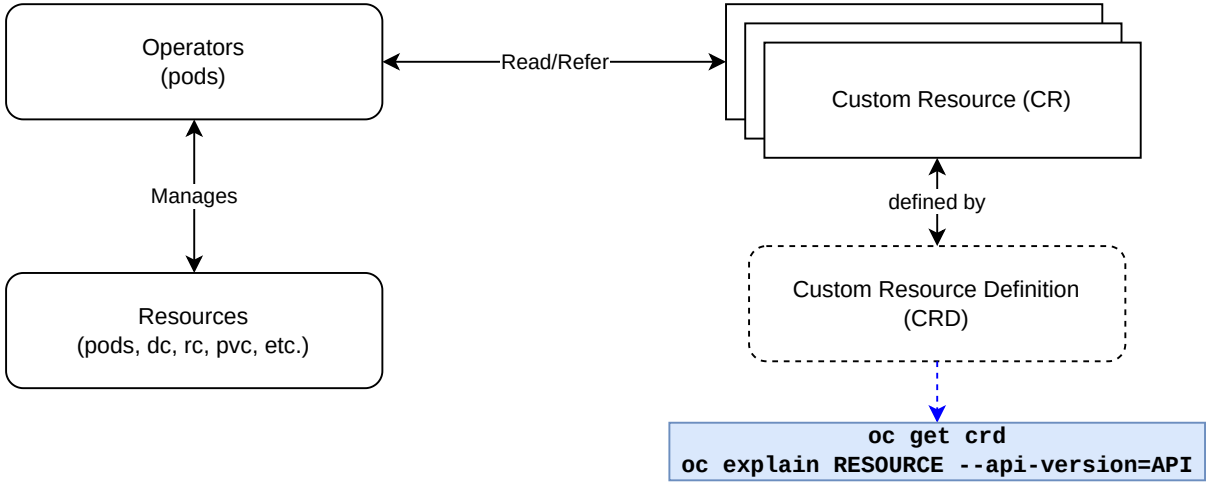

```
$ oc get oauth cluster -o yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
  ...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: htpasswd-secret
    mappingMethod: claim
    name: htpasswd_provider
    type: HTPasswd
```

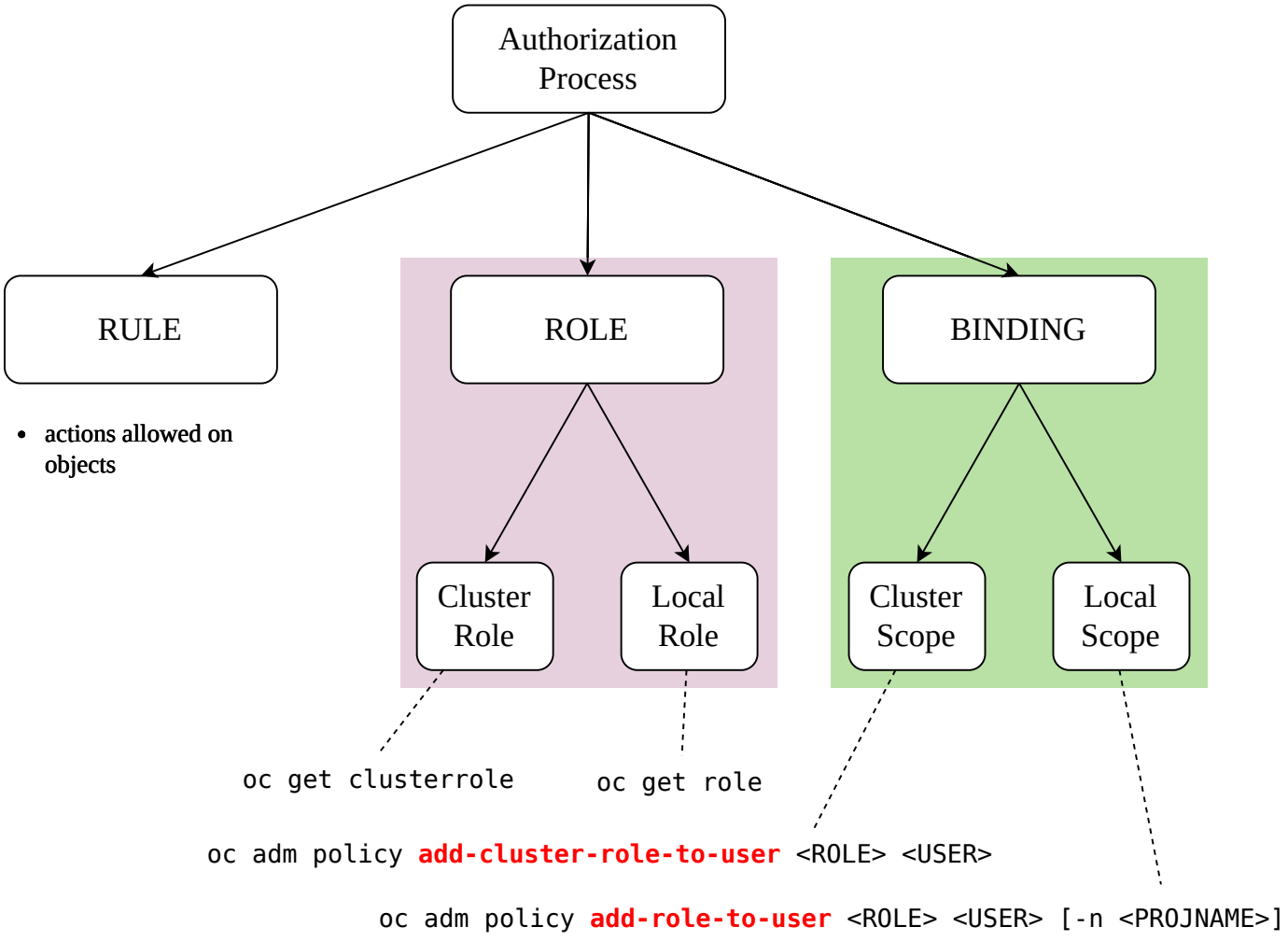
```
$ oc get secret htpasswd-secret -n openshift-config -o yaml
apiVersion: v1
data:
  htpasswd: YWRT...
kind: Secret
metadata:
  name: htpasswd-secret
  namespace: openshift-config
  resourceVersion: "90940"
type: Opaque
```


OPENSHIFT CLUSTER (DO380-Trusted TLS Certs)
1
oc create configmap <CONFIGMAP> --from-file ca-bundle.crt=<CERTIFICATE> -n openshift-config
Namespace: openshift-config
ConfigMap: combined-certs
data
combined-cert.pem
Secret: custom-tls
data
: combined-cert.pem
: wildcard-api-key.pem
cluster
trustedCA
combined-certs
2
oc patch proxy/cluster --type=merge --patch='{"spec":{"trustedCA":{"name":"<CONFIGMAP_NAME>"}}}'
3
oc create secret tls <SECRET_NAME> --cert <CERTIFICATE> --key <KEY> -n <NAMESPACE>
Namespace: openshift-ingress
Secret: custom-tls-bundle
data
: combined-cert.pem
: wildcard-api-key.pem
Namespace: openshift-ingress-operator
ingresscontrollers.operator: default
defaultCertificate
custom-tls-bundle
cluster
servingCerts
api.ocp4.example.com
custom-tls
4
oc patch ingresscontrollers.operator/default --type=merge --patch='{"spec":{"defaultCertificate":{"name":"<SECRET>"}}}'
-n openshift-ingress-operator
5
oc patch apiserver cluster --type=merge --patch='{"spec":{"servingCerts":{"namedCertificates":"[{"names":["<API-SERVER-URL>"],"servingCertificate":
{"name":"<SECRET-NAME>"}}]}}}'

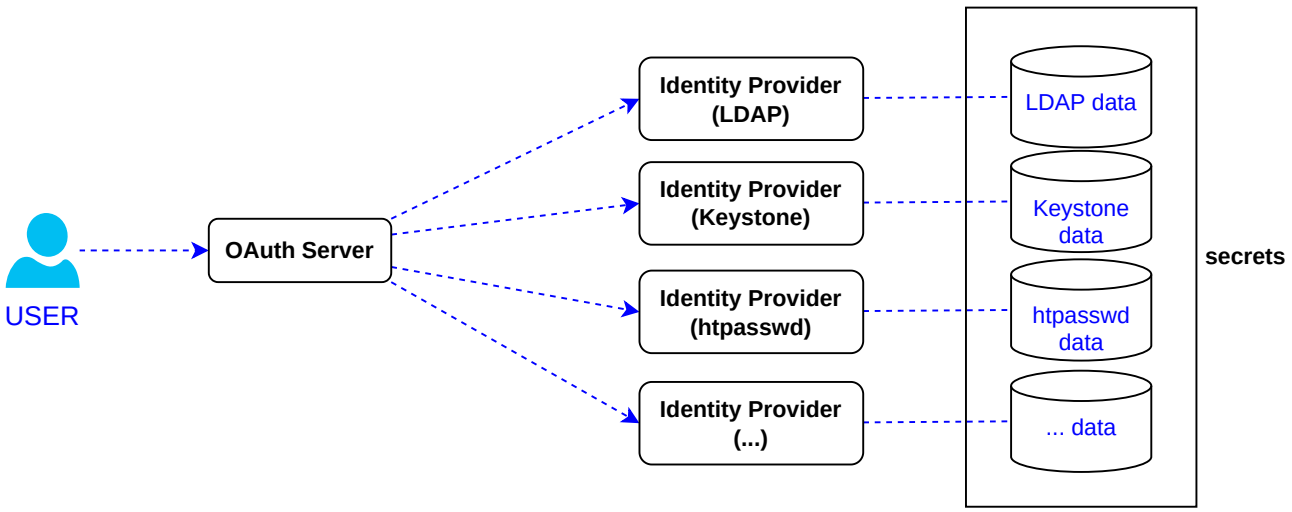


An operator is a control loop program and it can respond to events.
It communicates with the API server to manage k8s resources



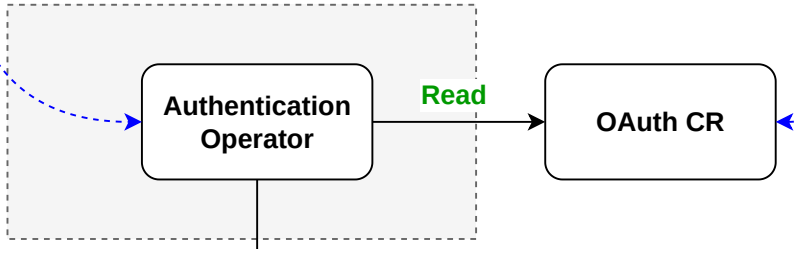


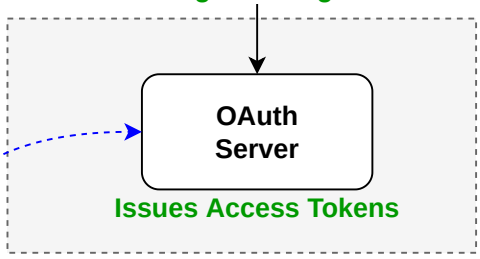
- actions allowed on objects




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

Project: *openshift-authentication-operator*

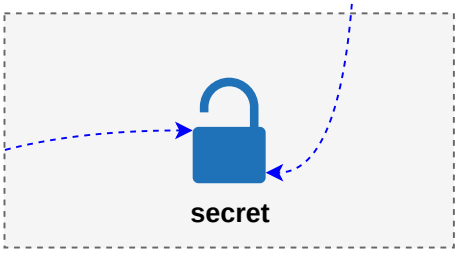




Project: *openshift-authentication*

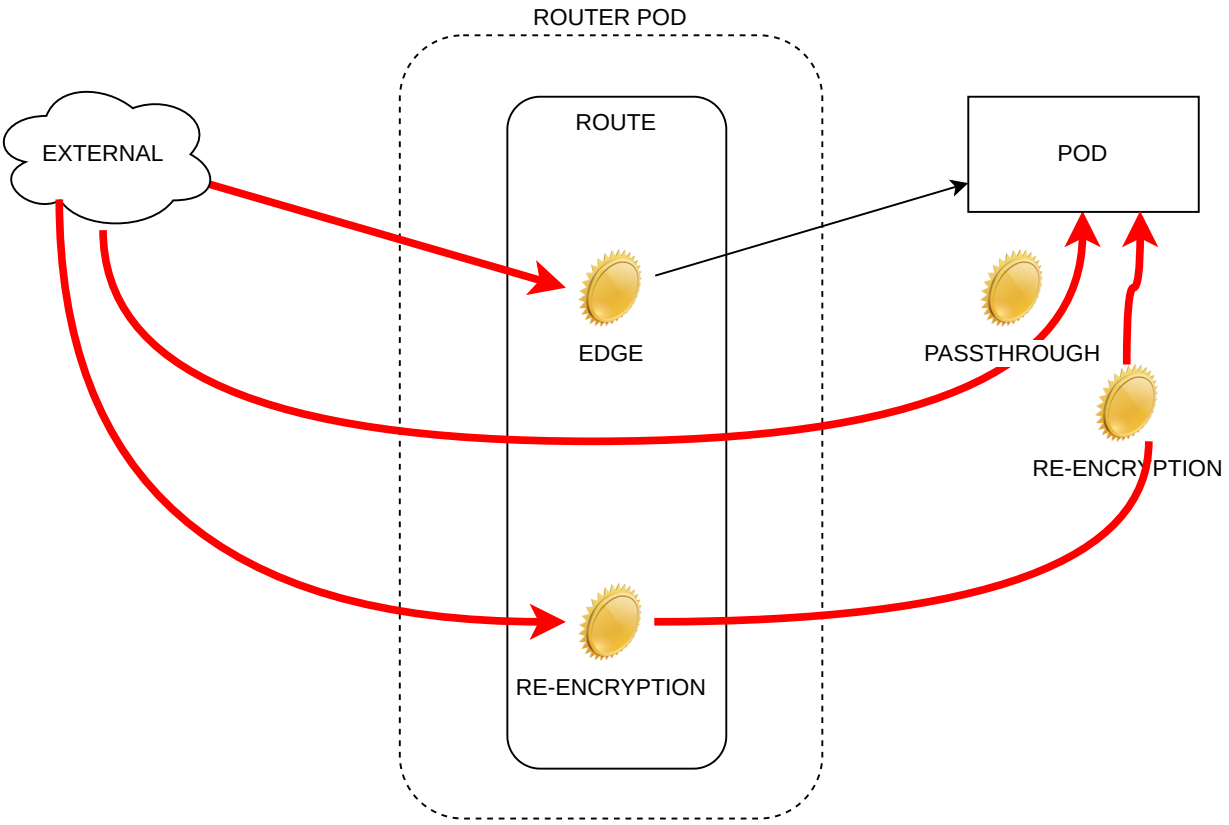
```
$ oc get pods -n openshift-authentication
```

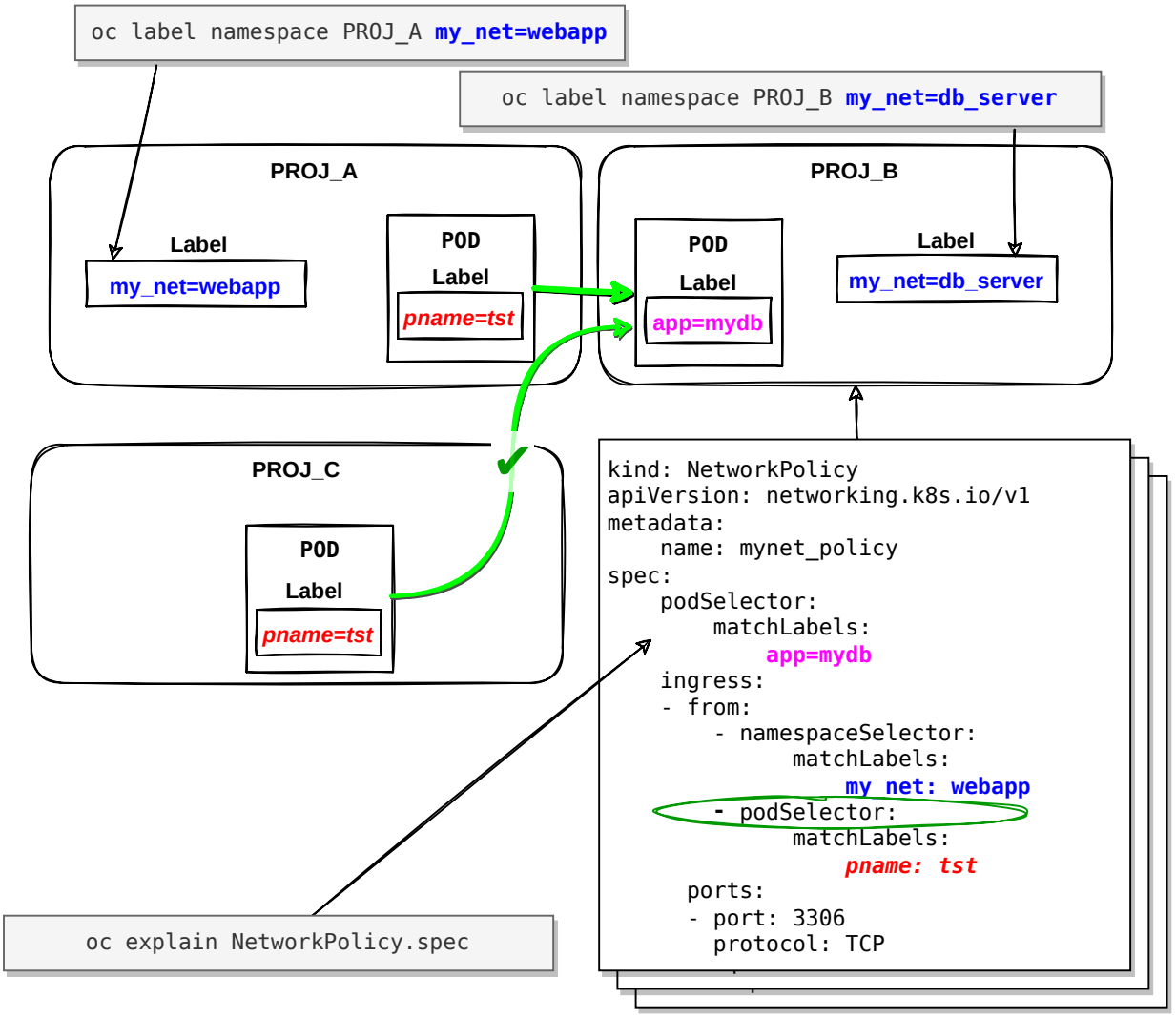
```
$ oc get oauth cluster -o yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
  ...
spec:
  identityProviders:
  - htpasswd:
      fileData:
        name: htpasswd-secret
    mappingMethod: claim
    name: htpasswd_provider
    type: HTPasswd
```



Project: *openshift-config*

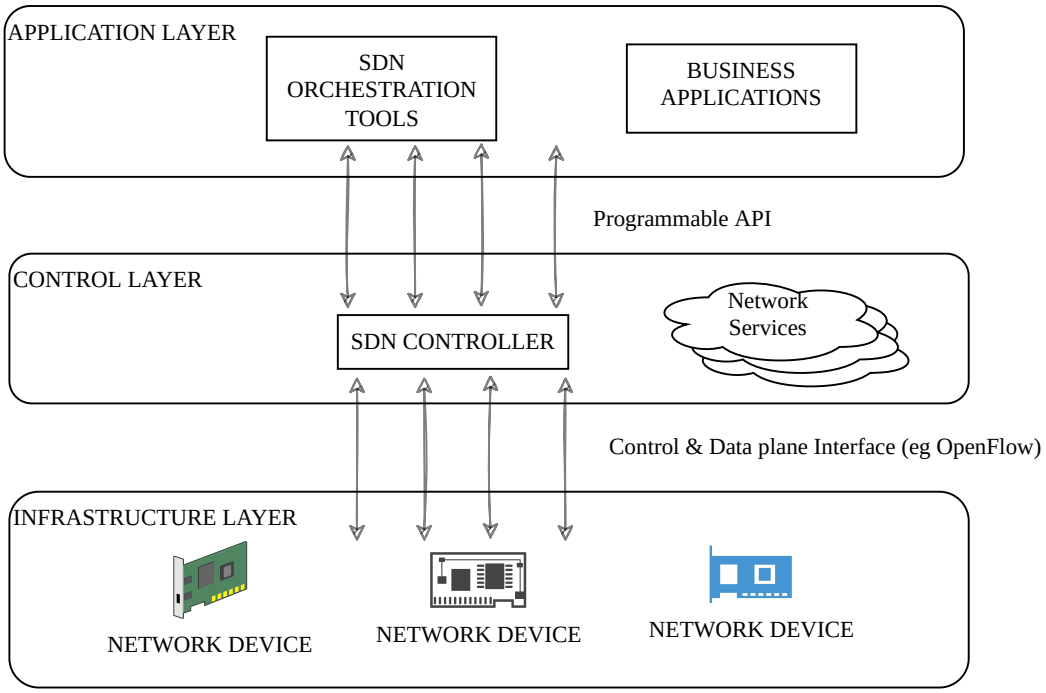
```
$ htpasswd -cBb ./myusers <USER> <PASSWORD>
$ oc create secret generic htpasswd-secret --from-file htpasswd=./myhtpasswd -n openshift-config
```


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

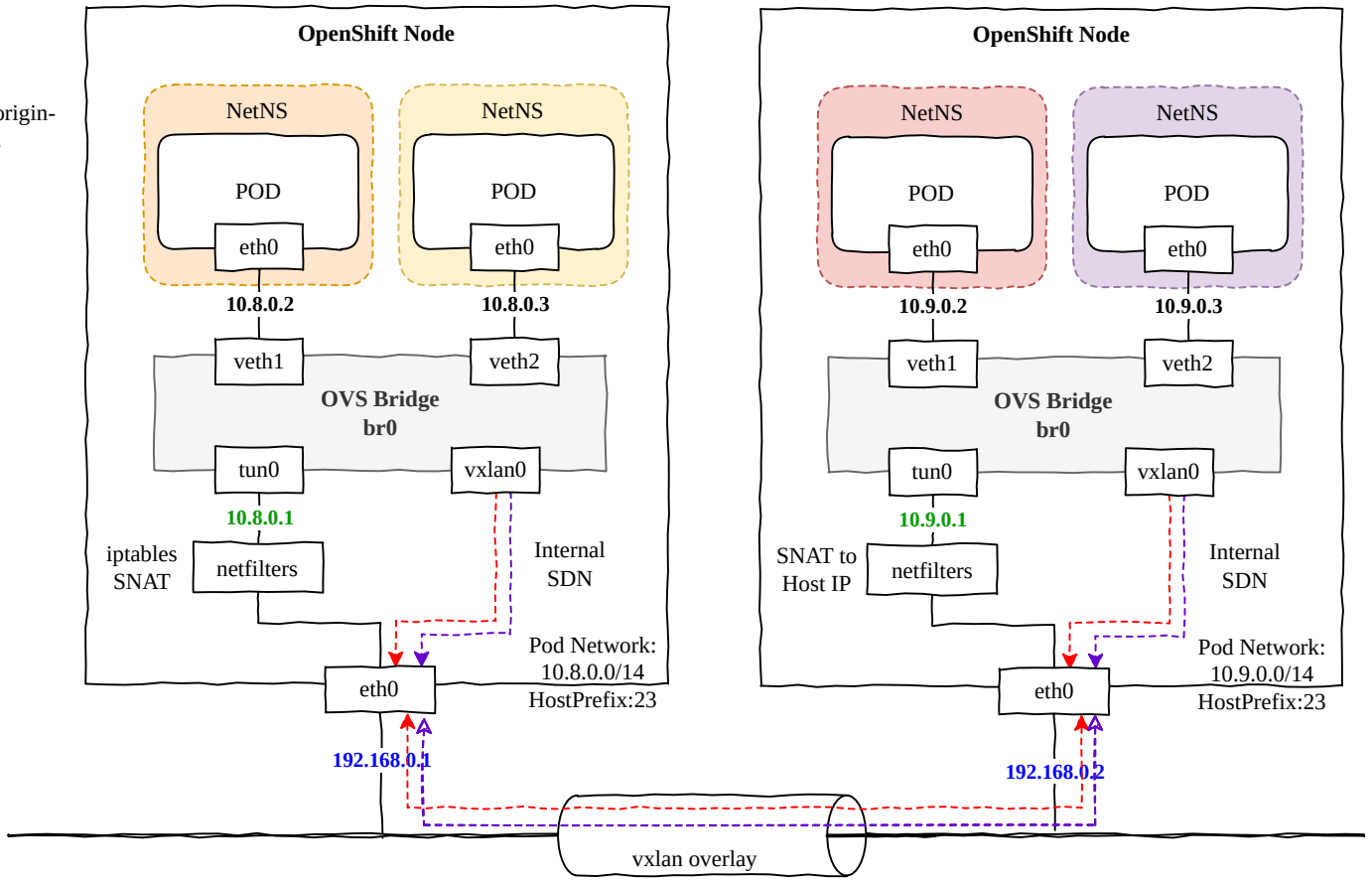

oc label namespace PROJ_A my_net=webapp
oc label namespace PROJ_B my_net=db_server
PROJ_A
Label
my_net=webapp
POD
Label
pname=tst
PROJ_B
POD
Label
app=mydb
Label
my_net=db_server
PROJ_C
POD
Label
pname=tst
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
name: mynet_policy
spec:
podSelector: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

SDN

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



use openshift/origin-tools



nnnnnnnn . nnnnnnxx . xxxxxxxx . xxxxxxxx

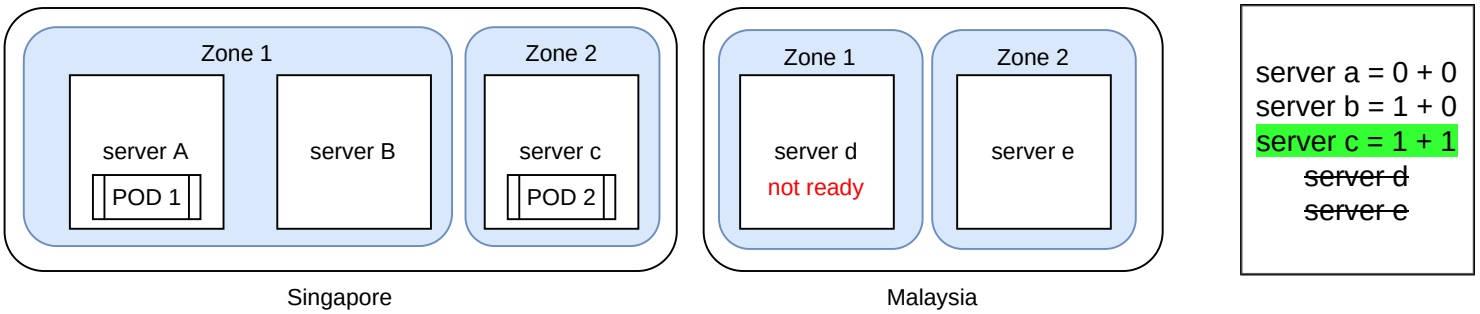
POD Scheduling

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.

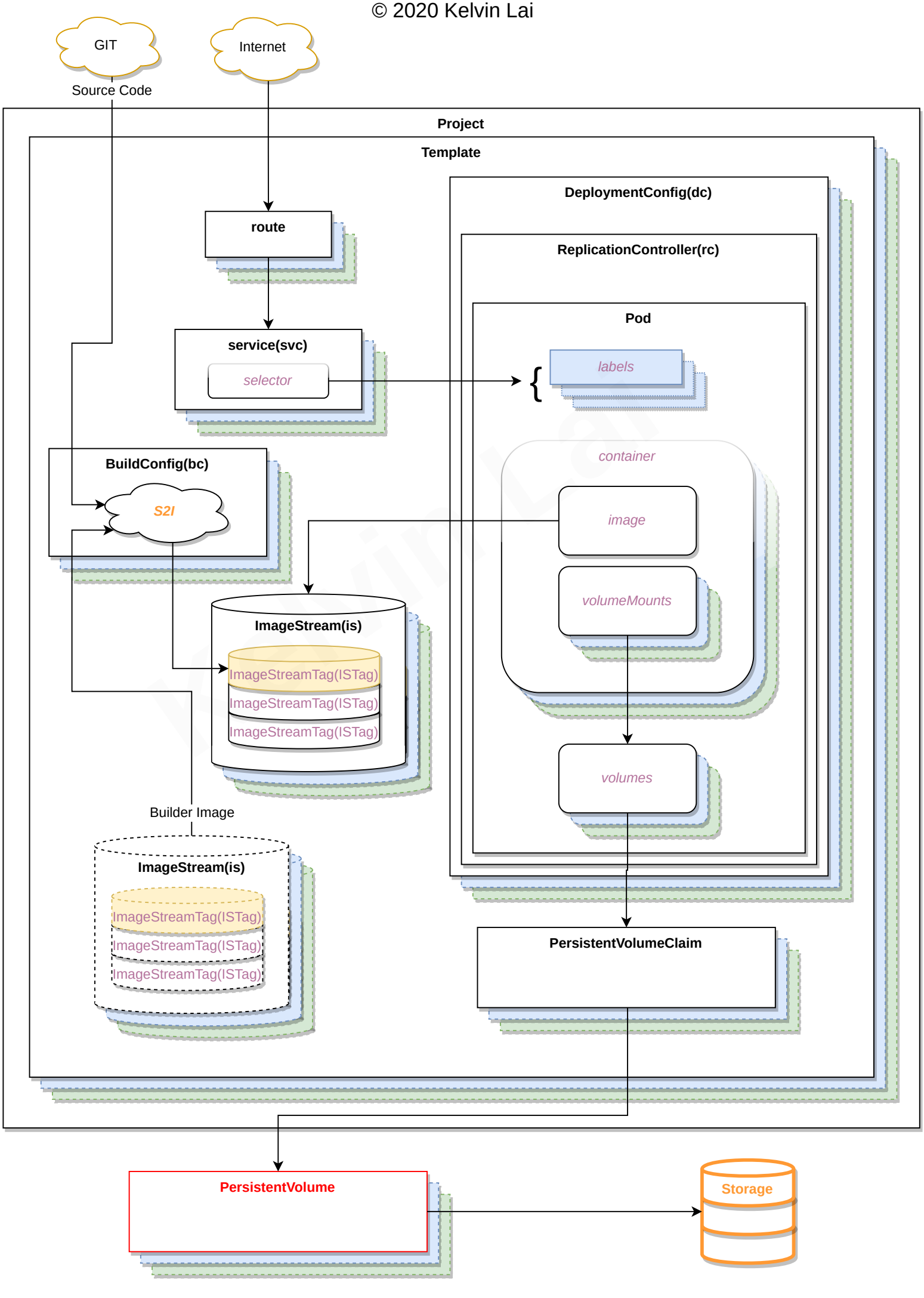
Zone (availability zone)

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

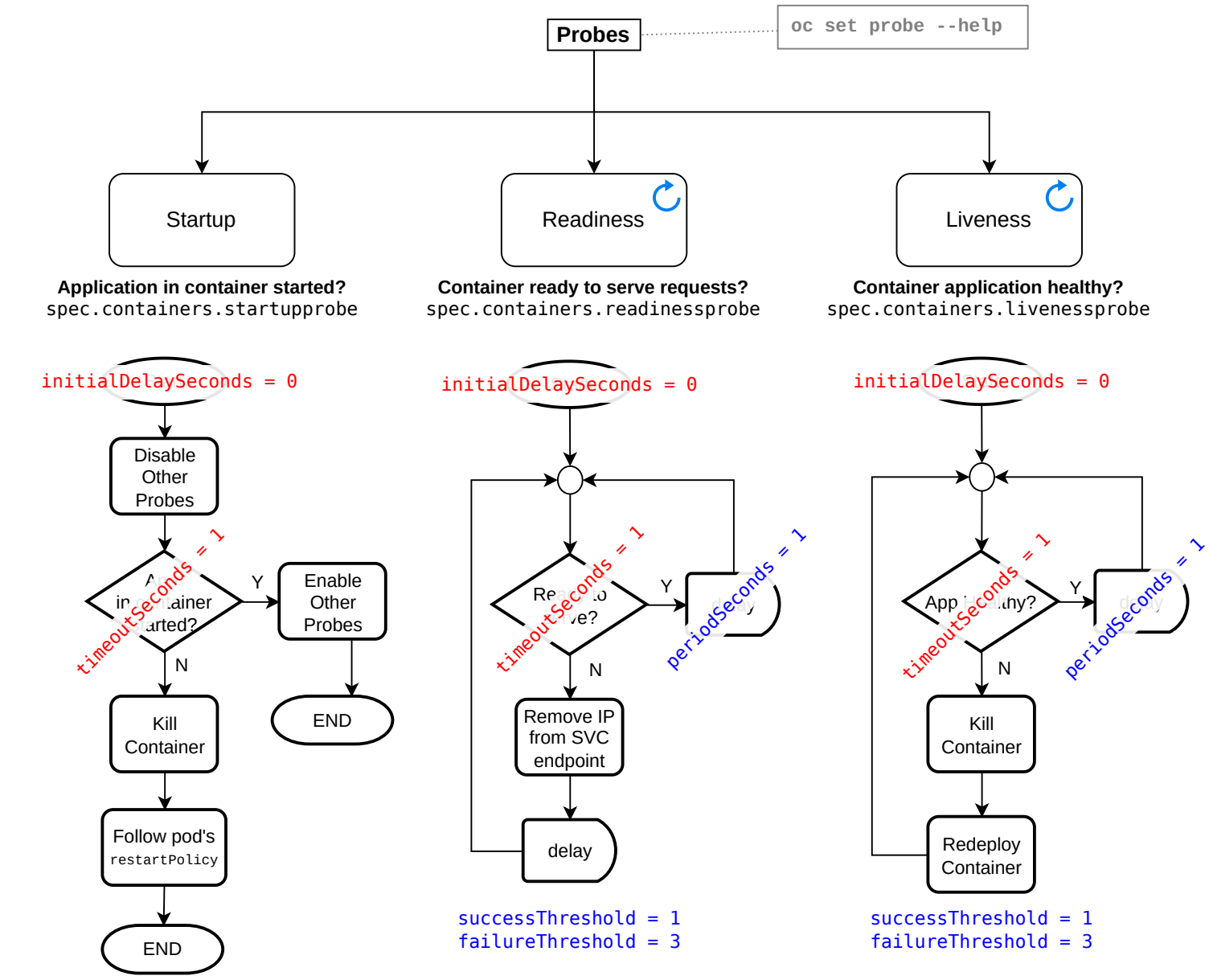
A set of hosts that share common critical infra components (ups, switch, storage)

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

OpenShift Resource Types



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

Blue-Green Deployment

Deploy production app, green

```
oc new-app registry.example.com/myapp:v1 --name green
oc expose deployment green       # expose dc/deployment to get svc
oc expose svc green              # expose svc to get route
oc get route                     # get hostname to access app
```

Deploy updated version of app, blue

```
oc new-app registry.example.com/myapp:v2 --name blue
oc expose deployment blue        # expose dc/deployment to get svc

oc patch route green -p '{"spec":{"to":{"name":"blue"}}}'      # update route to use blue svc
```

A/B Deployment

Deploy production app, app-a

```
oc new-app registry.example.com/myapp:v1 --name app-a
oc expose deployment app-a            # expose dc/deployment to get svc
oc expose svc app-a --name myroute    # expose svc to get route
oc get route                          # get hostname to access app
```

Deploy updated version of app, app-b

```
oc new-app registry.example.com/myapp:v2 --name app-b
oc expose deployment app-b       # expose dc/deployment to get svc
```

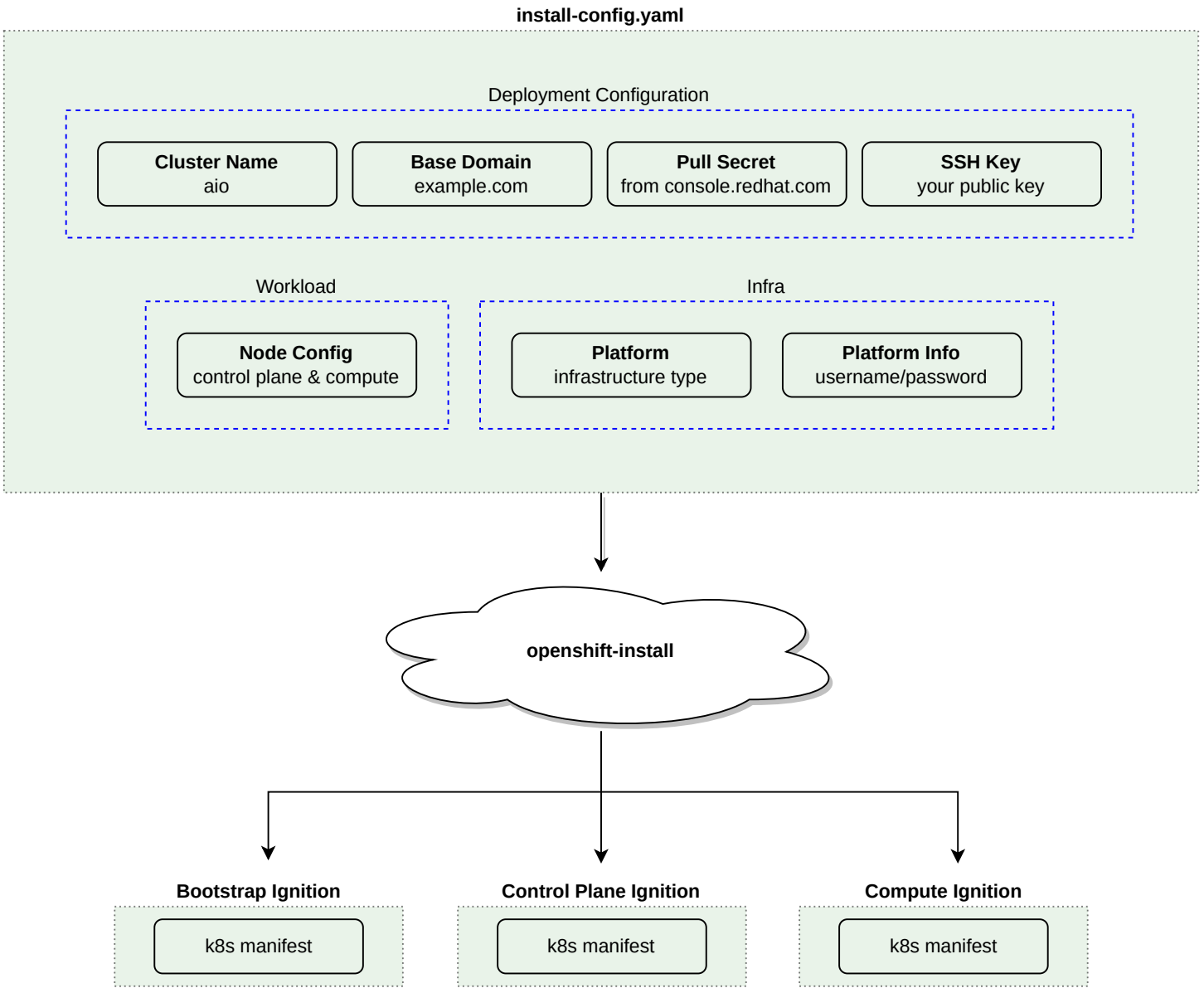
Note: router balances traffic according to weights(default=1).

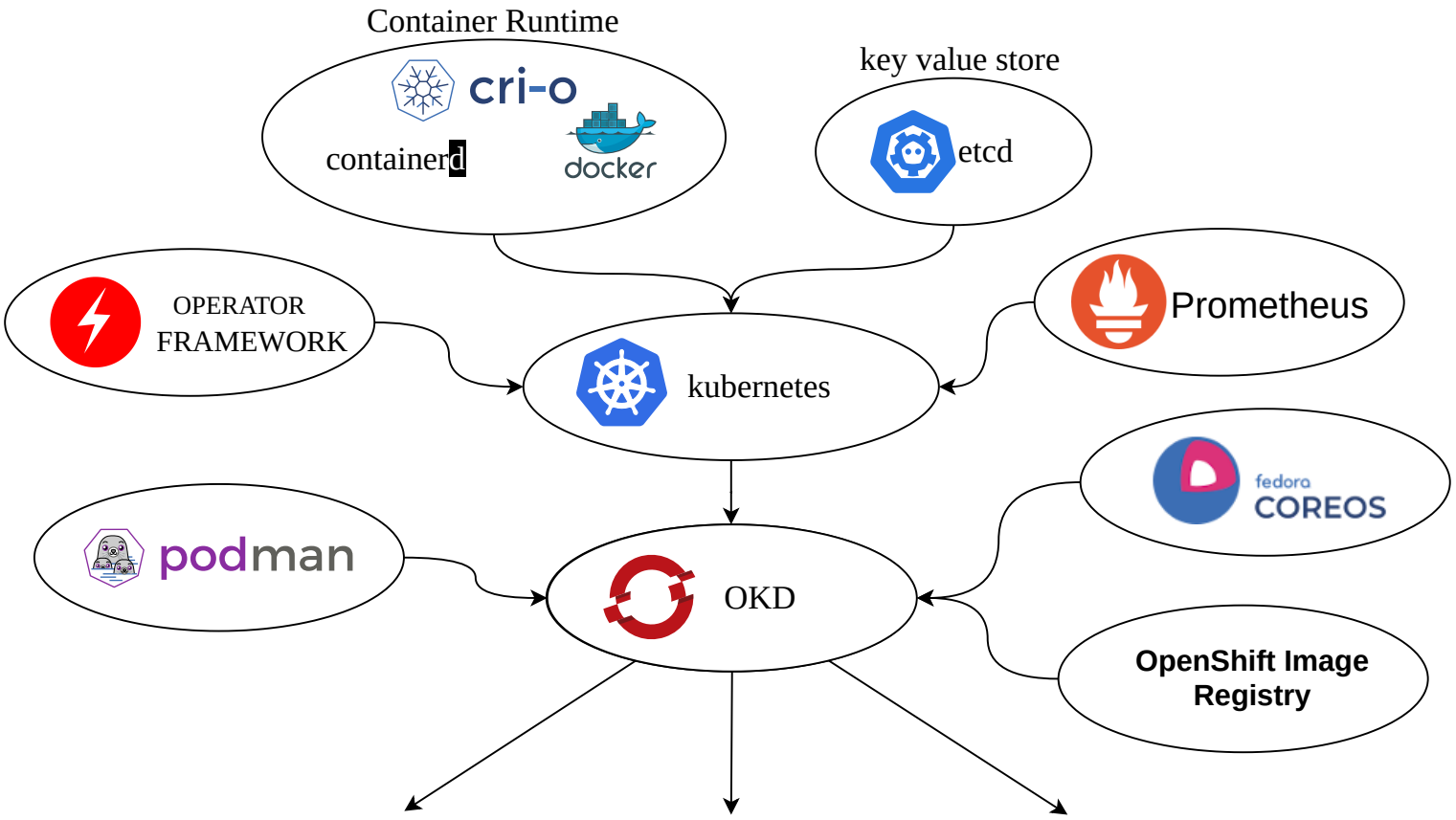
Add service app-b to route and set weightage as 80% for app-a and 20% for app-b

```
oc set route-backends myroute app-a=80 app-b=20
oc set route-backends myroute    # verify configuration
```

N-1 Compatibility & Graceful Termination - Refer to Notes

find out haproxy weight pct vs conn


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



Red Hat OpenShift Container Platform

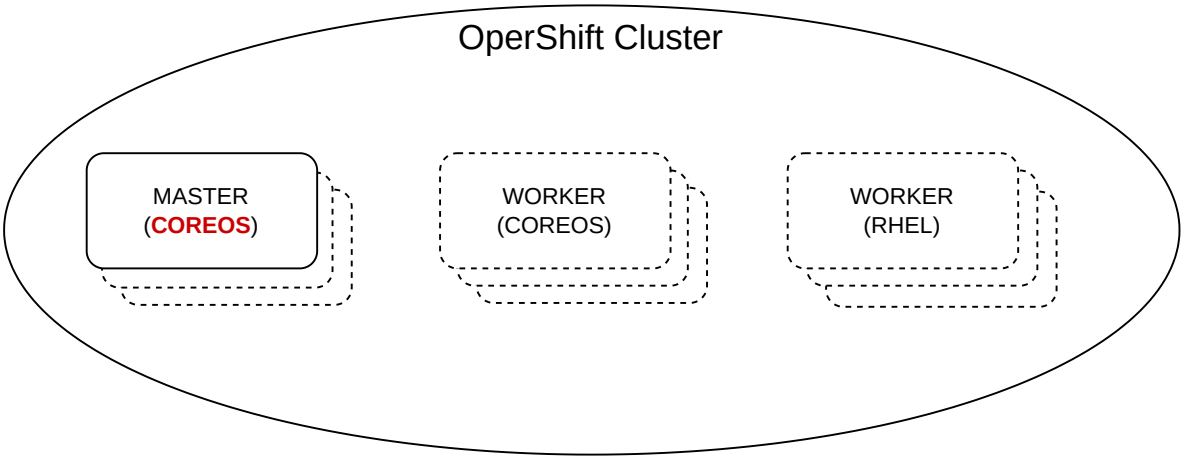
- Public/private DC.
- Bare metal and multiple cloud and virtualization providers.
- Full control by customer.

Red Hat OpenShift Dedicated

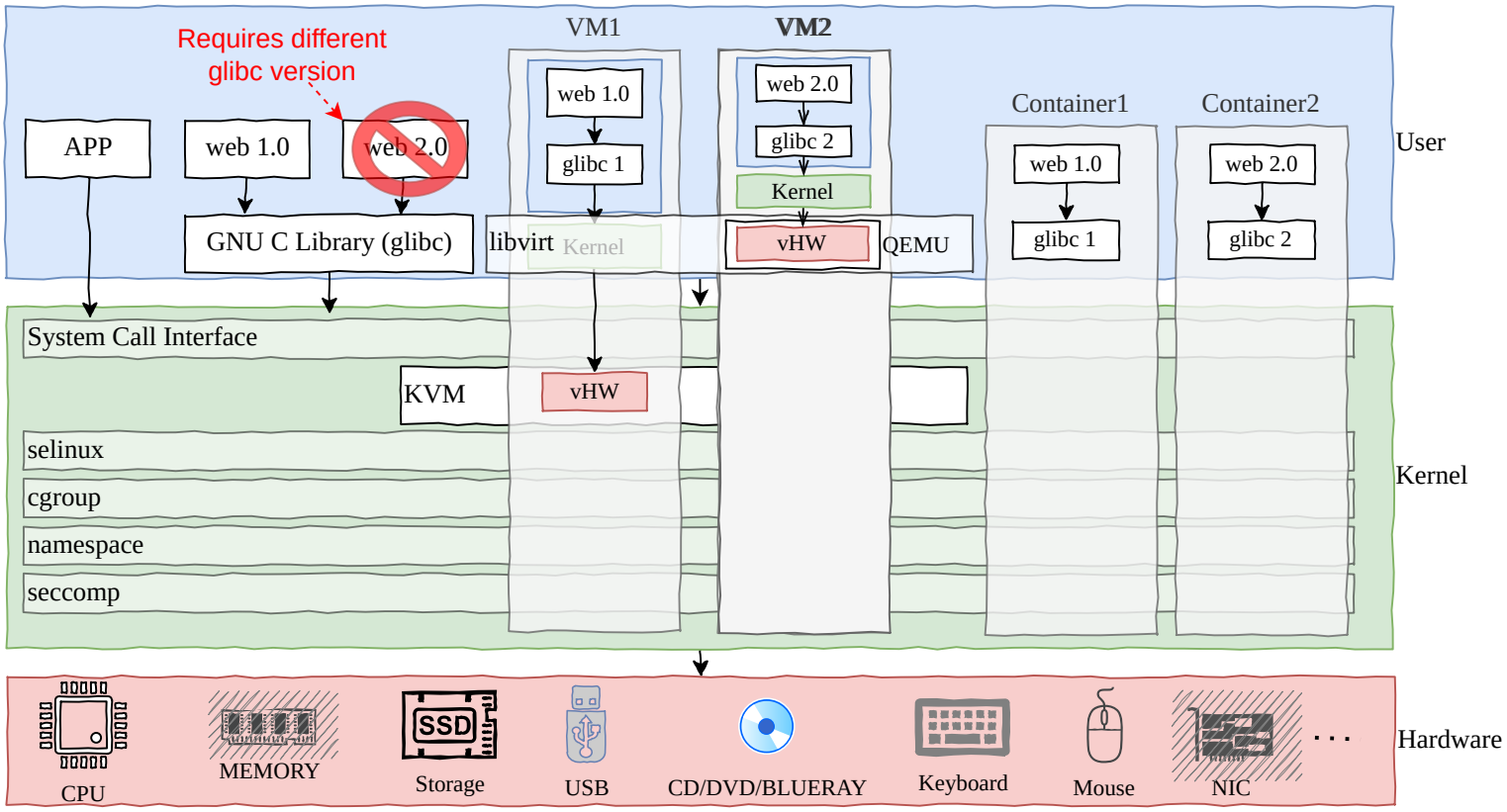
- Managed cluster in public cloud.
- RH manages the cluster.
- Customer manages updates and add-on services.

Red Hat OpenShift Online

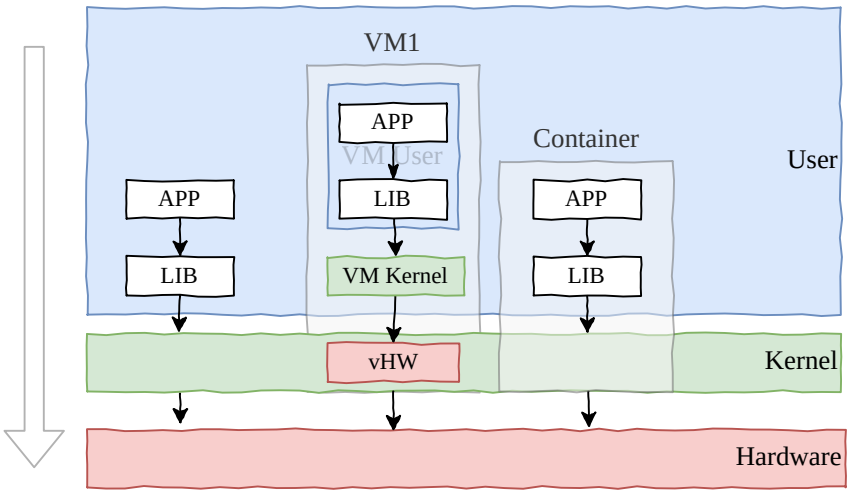
- Public hosted cluster.
- Shared resources by multiple customers.
- RH manages cluster life cycle.



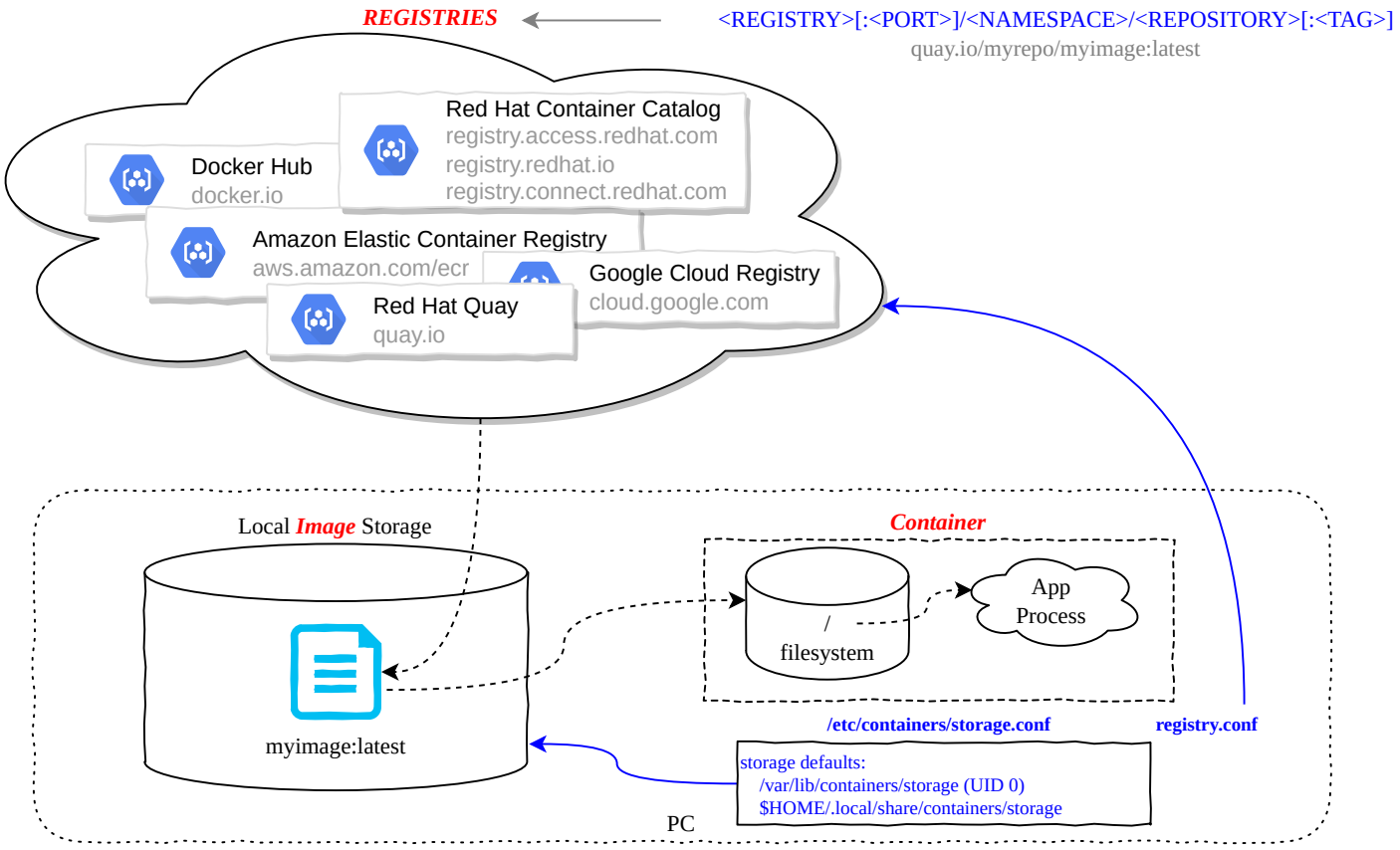
VM vs Container

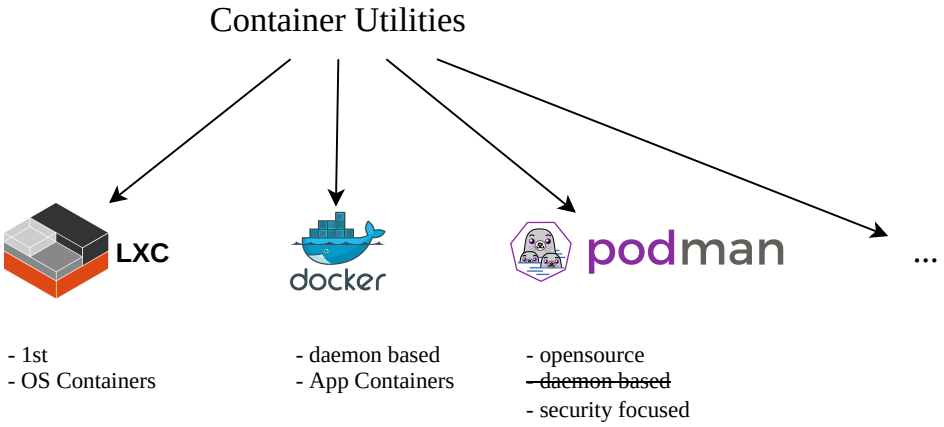


Ref: https://www.redhat.com/en/blog/all-you-need-know-about-kvm-userspace
https://www.packetcoders.io/what-is-the-difference-between-qemu-and-kvm/



Container Architecture

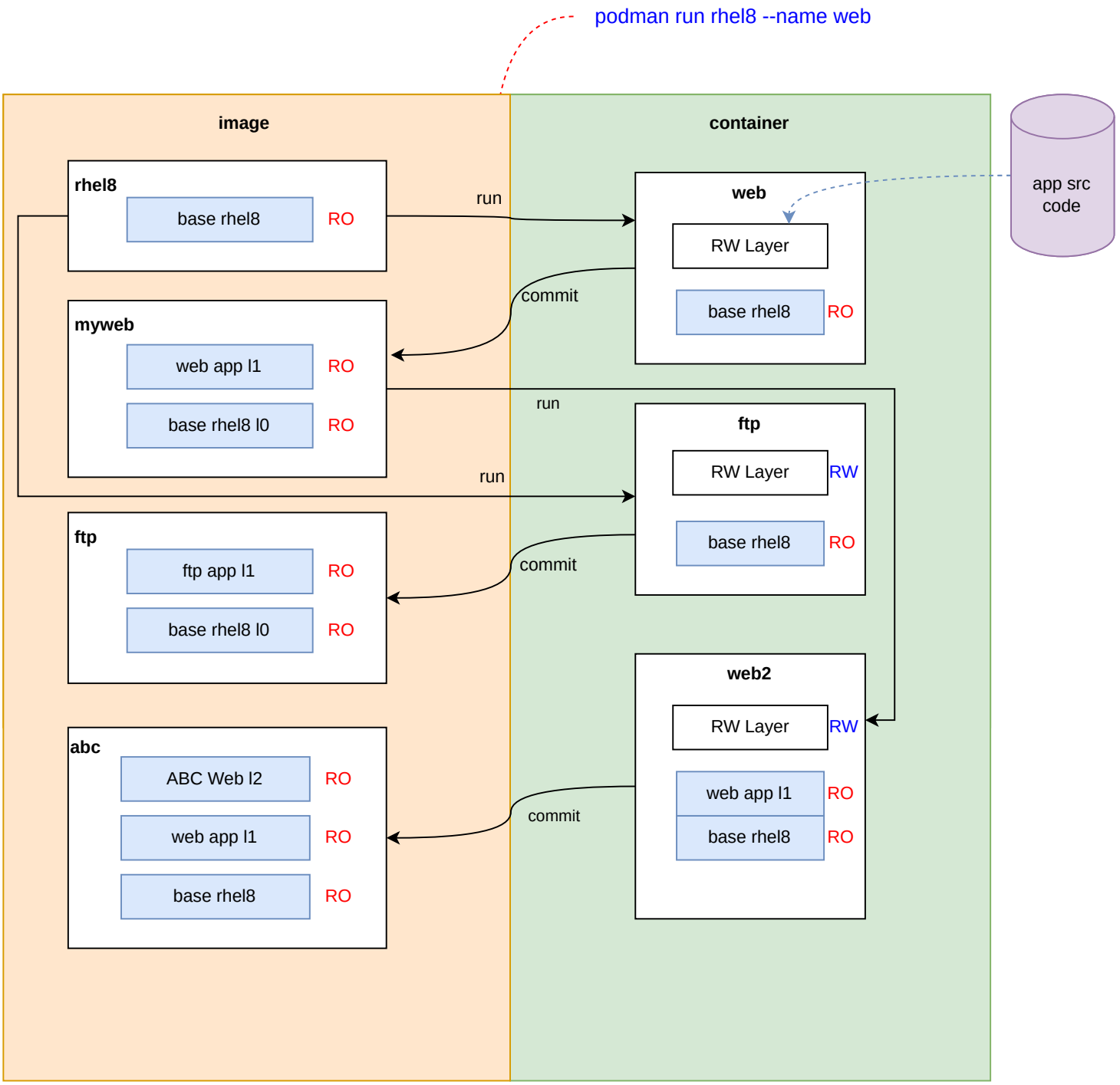




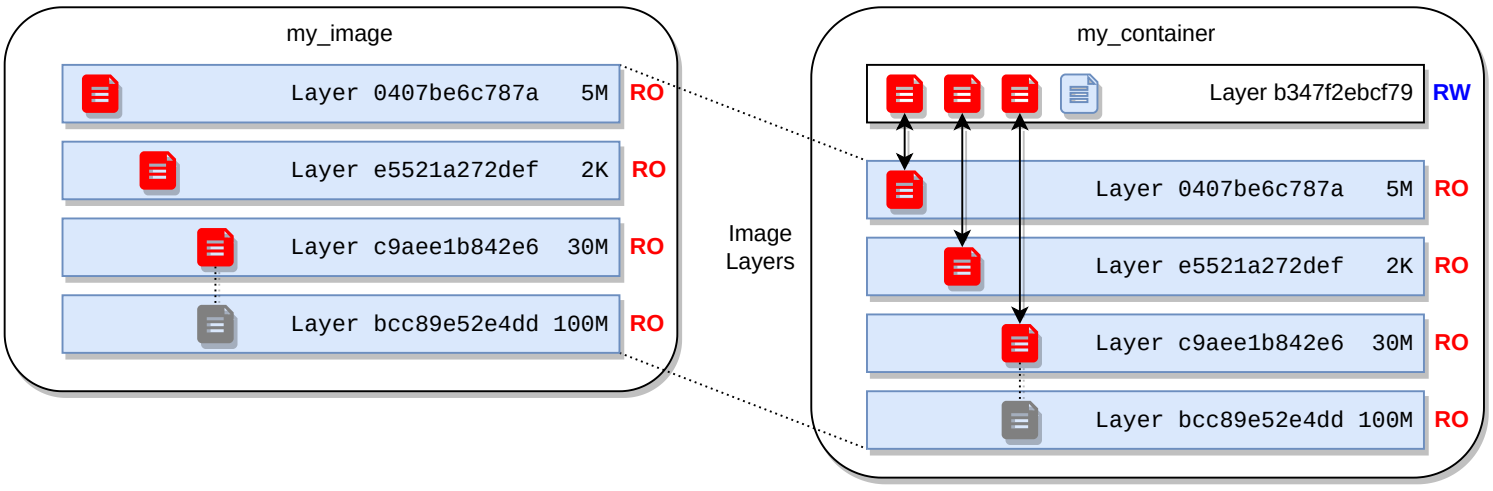
OS Container Vs Application Containers

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

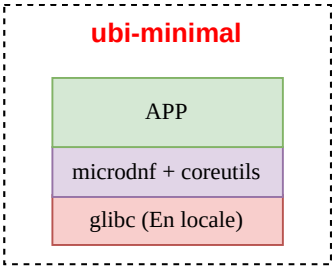


UnionFS - A Stackable Unification File System



BASE IMAGE TYPES

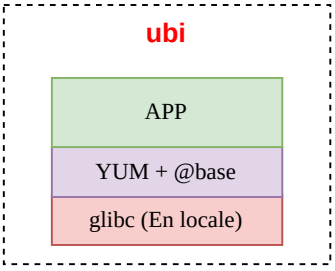
MINIMAL



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)

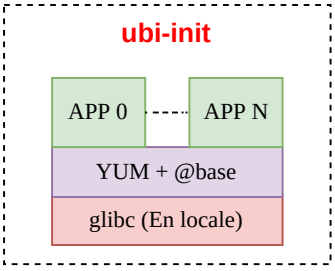
PLATFORM



For any apps that runs on RHEL

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

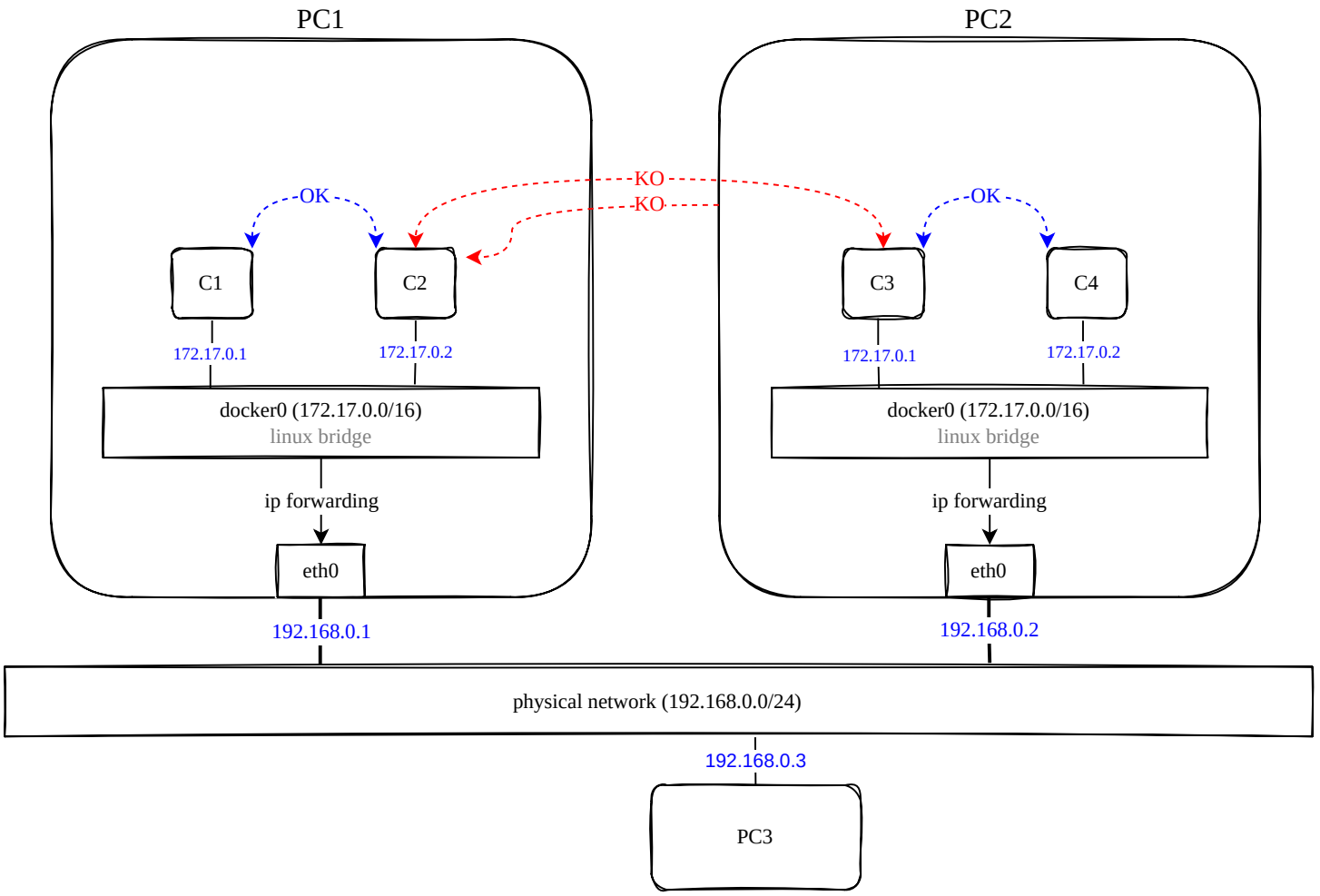
MULTI-SERVICE

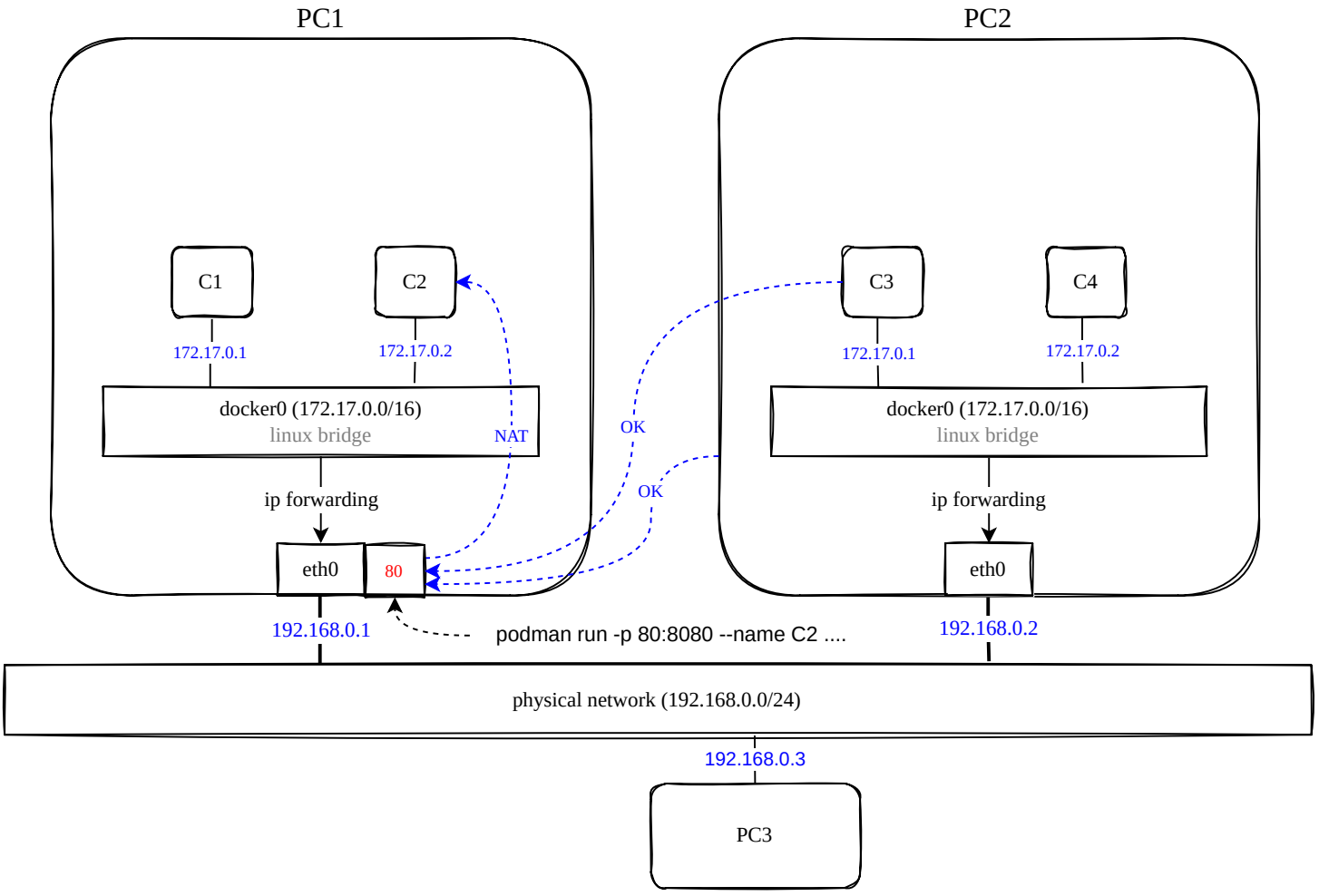


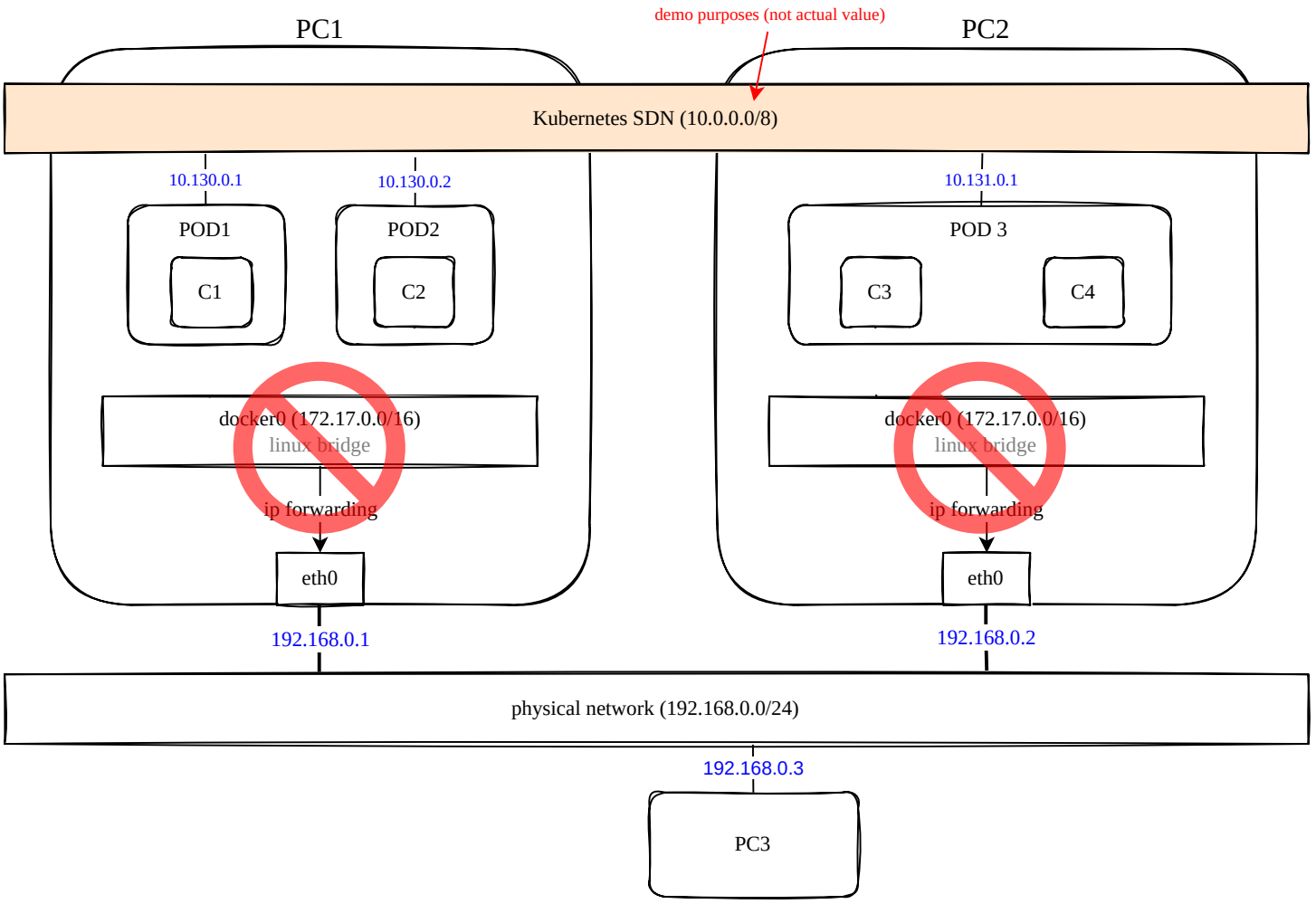
Eases running multi-service in single container

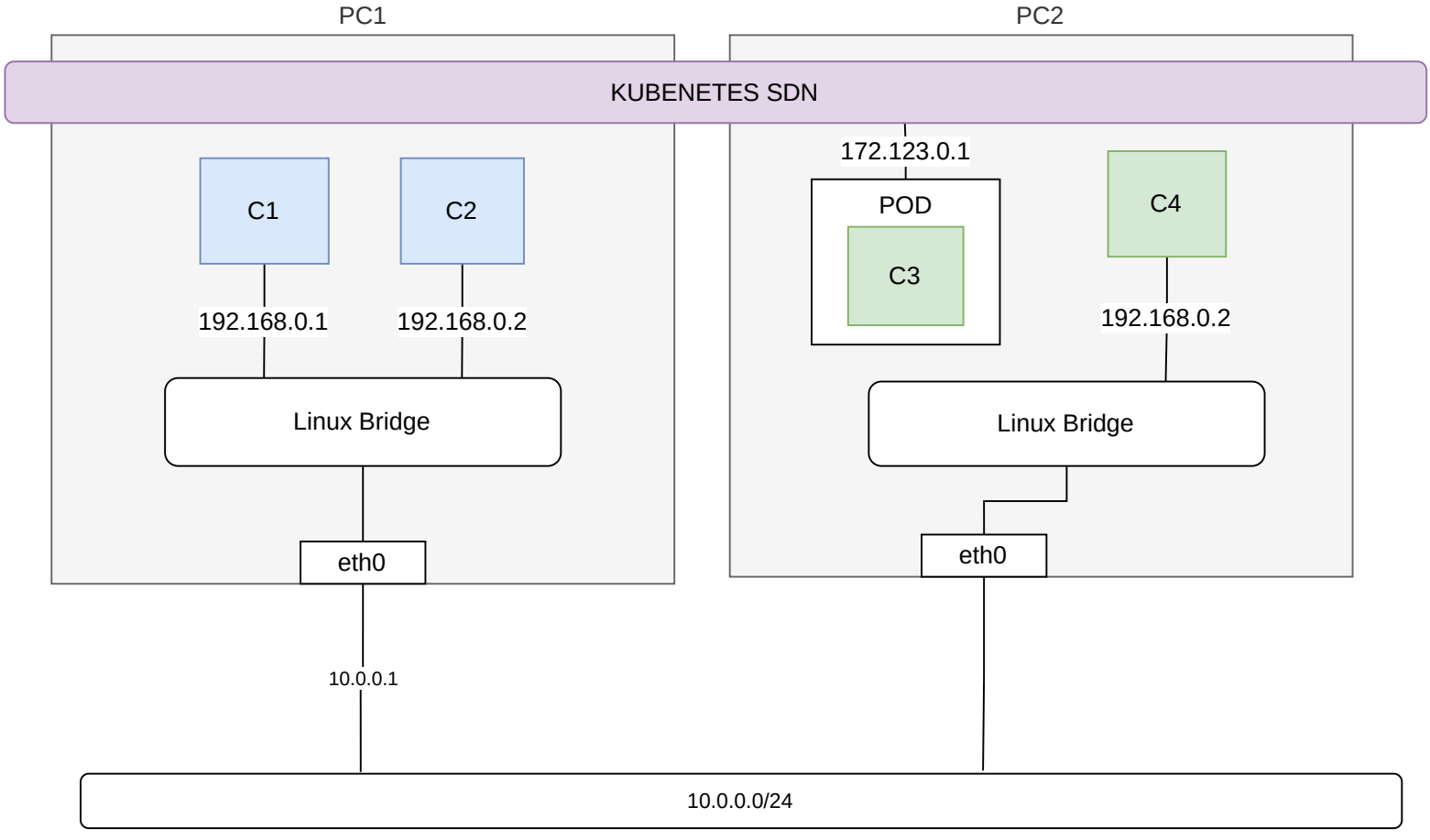
- configured to run systemd on start
- allows you to enable th services at build time

Basic Network - Container vs Kubernetes

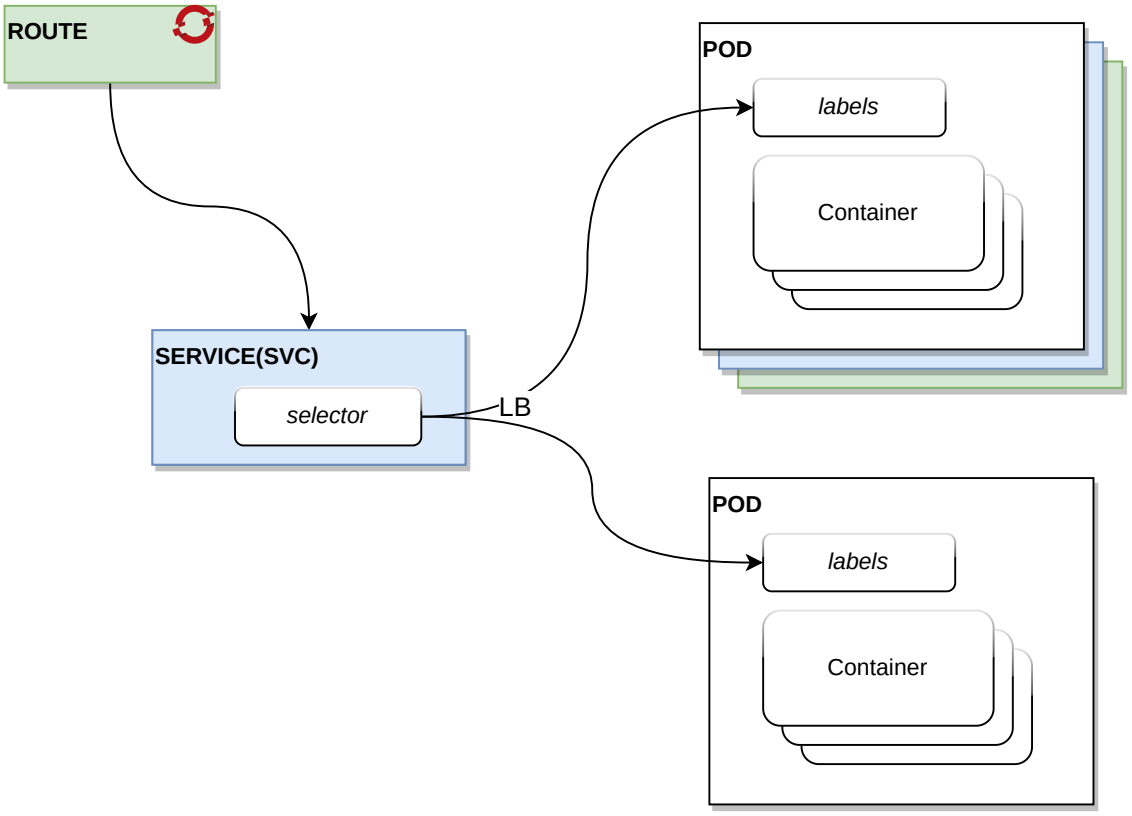





PC1
demo purposes (not actual value)
PC2
Kubernetes SDN (10.0.0.0/8)
10.130.0.1
10.130.0.2
10.131.0.1
POD1
POD2
POD 3
C1
C2
C3
C4
docker0 (172.17.0.0/16)
linux bridge
ip forwarding
eth0
192.168.0.1
192.168.0.2
physical network (192.168.0.0/24)
192.168.0.3
PC3


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

Route, Service and Pod Relationship



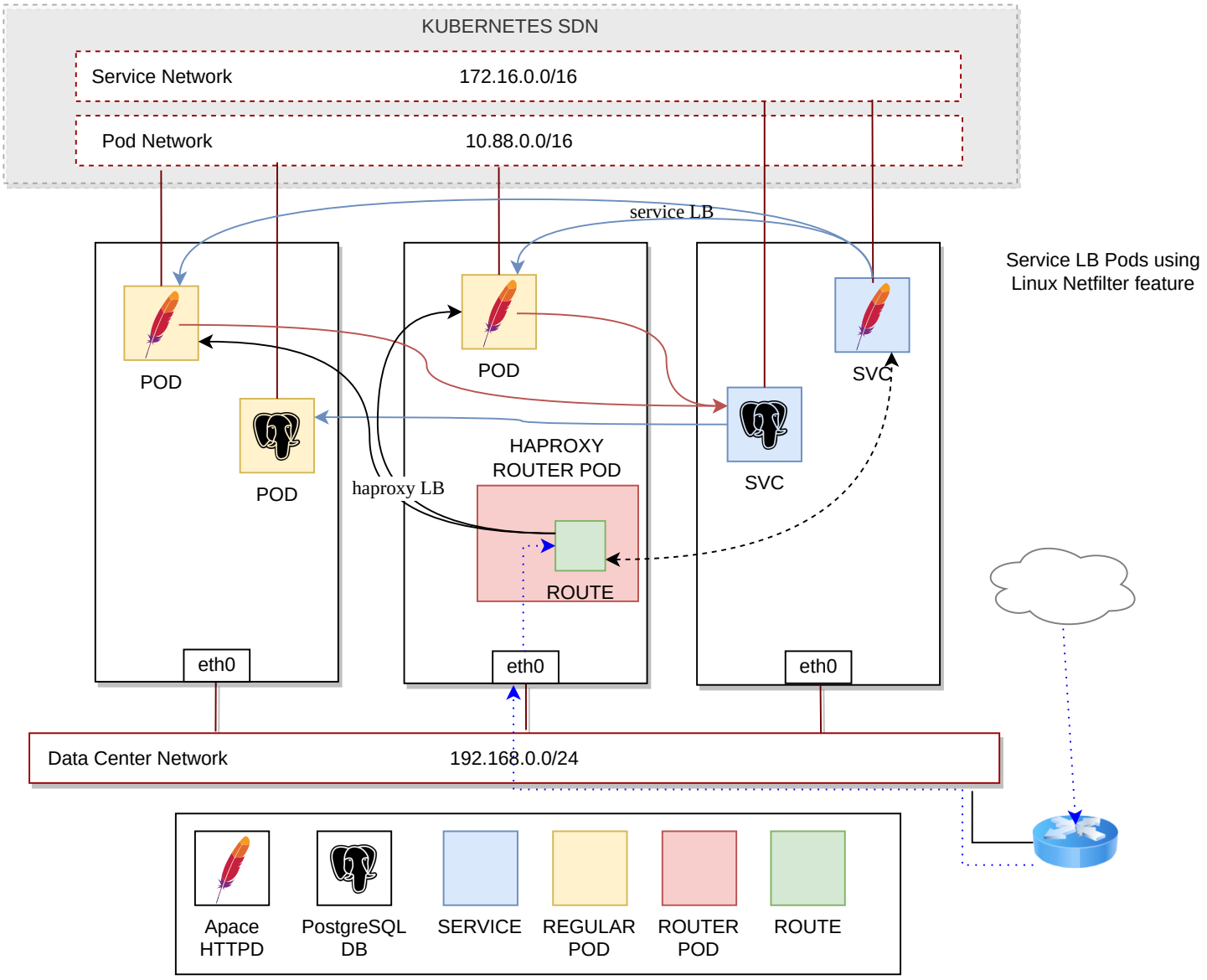
POD
A pod contains one or more containers.

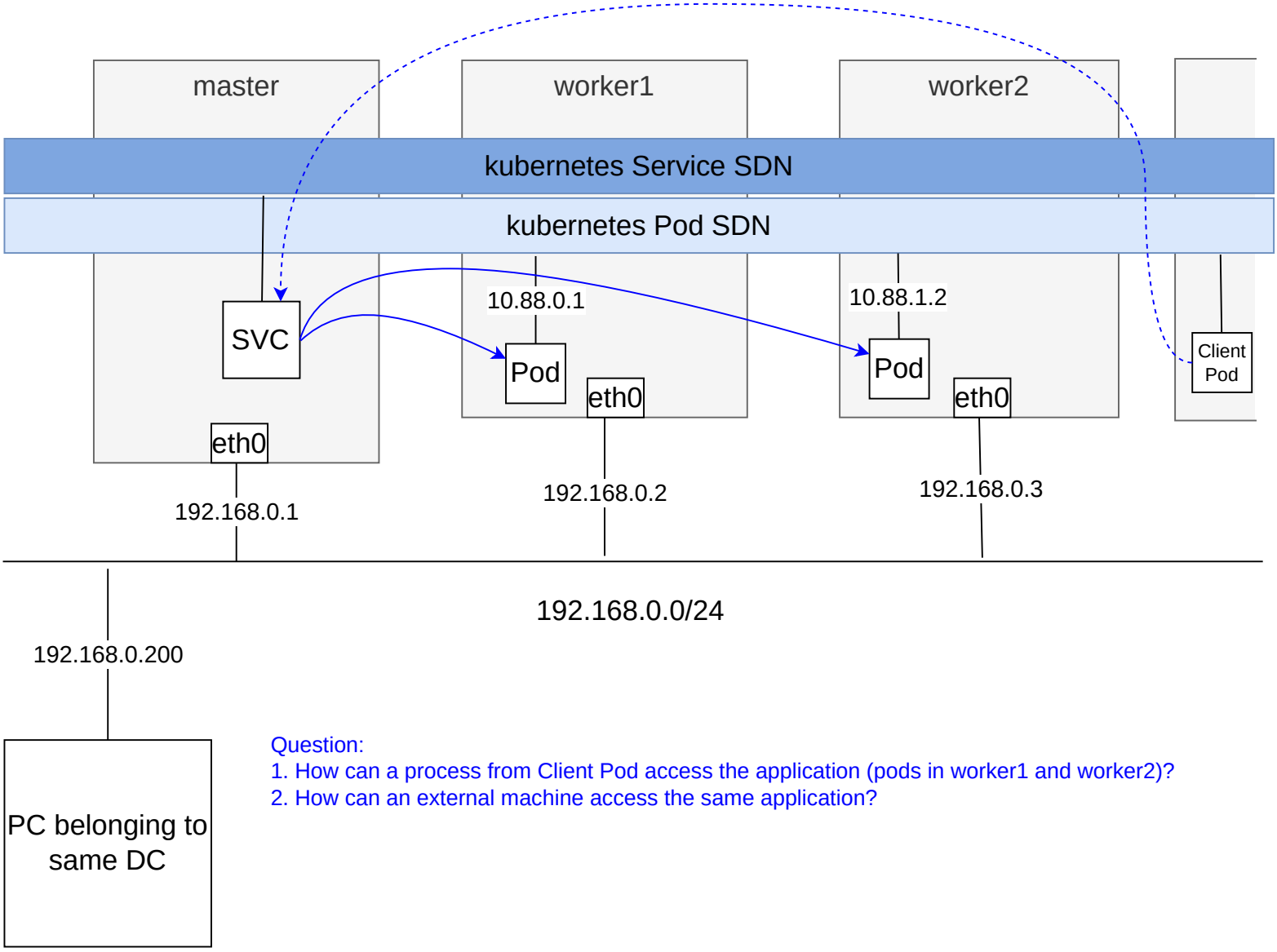
SERVICE
A service references the pod(s) by using the label selector.
The service load balances the connections between all the pods.

ROUTE
A route exposes the service to the external world.

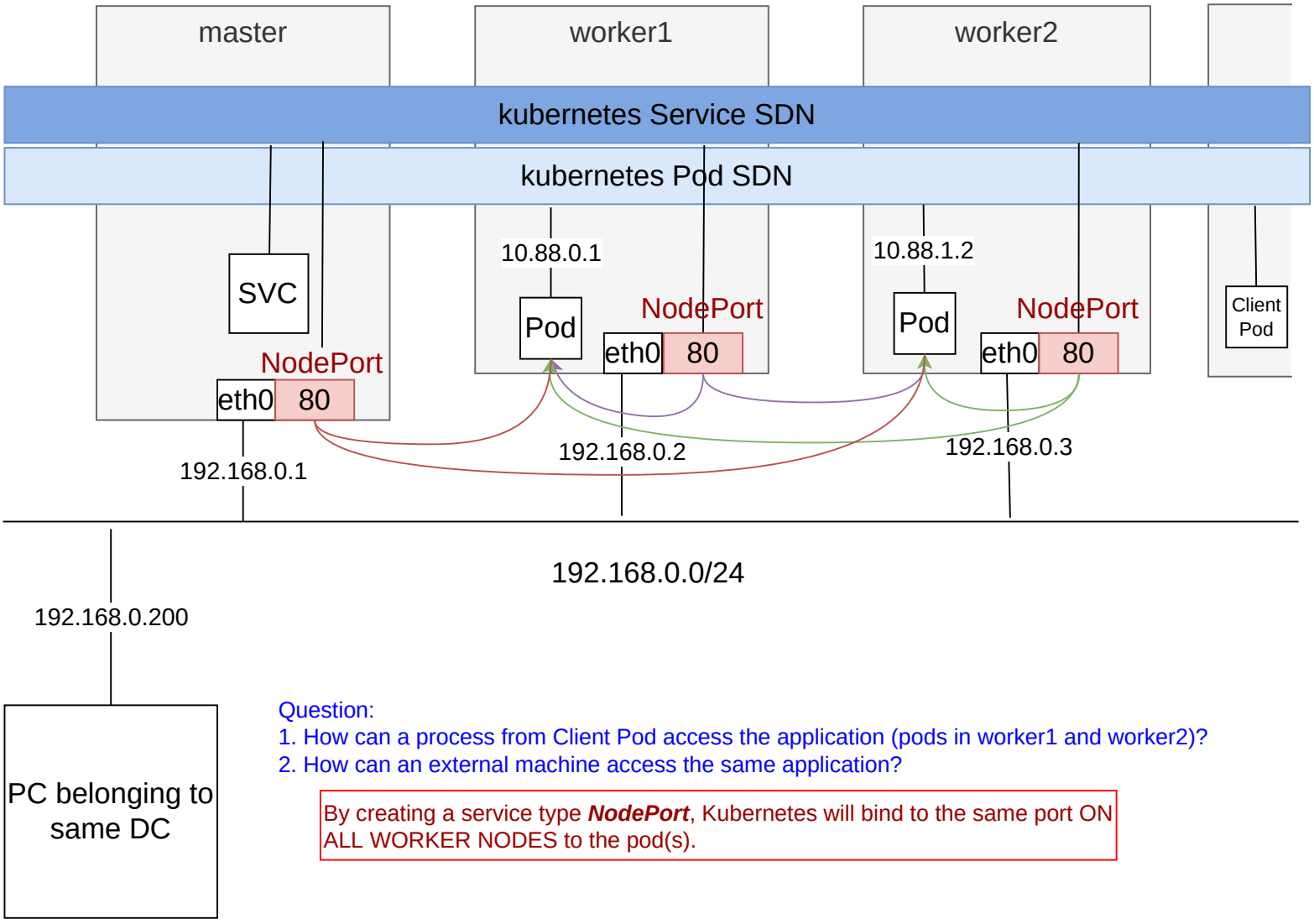
Warning: A service "can" refer to different pods, if the pods have the same label.

Sample of how Services are used





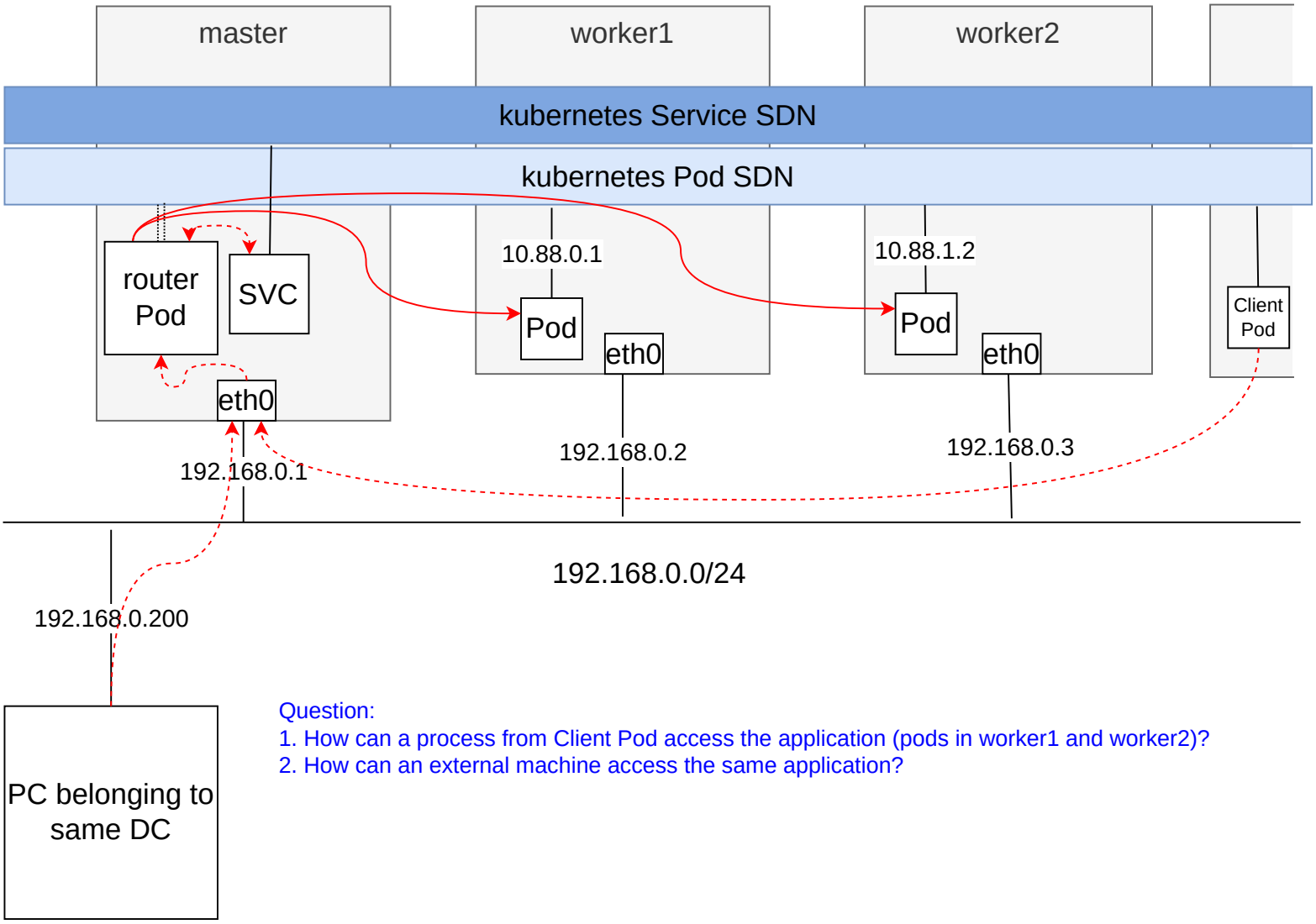
Question:
1. How can a process from Client Pod access the application (pods in worker1 and worker2)?
2. How can an external machine access the same application?



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?

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>
```

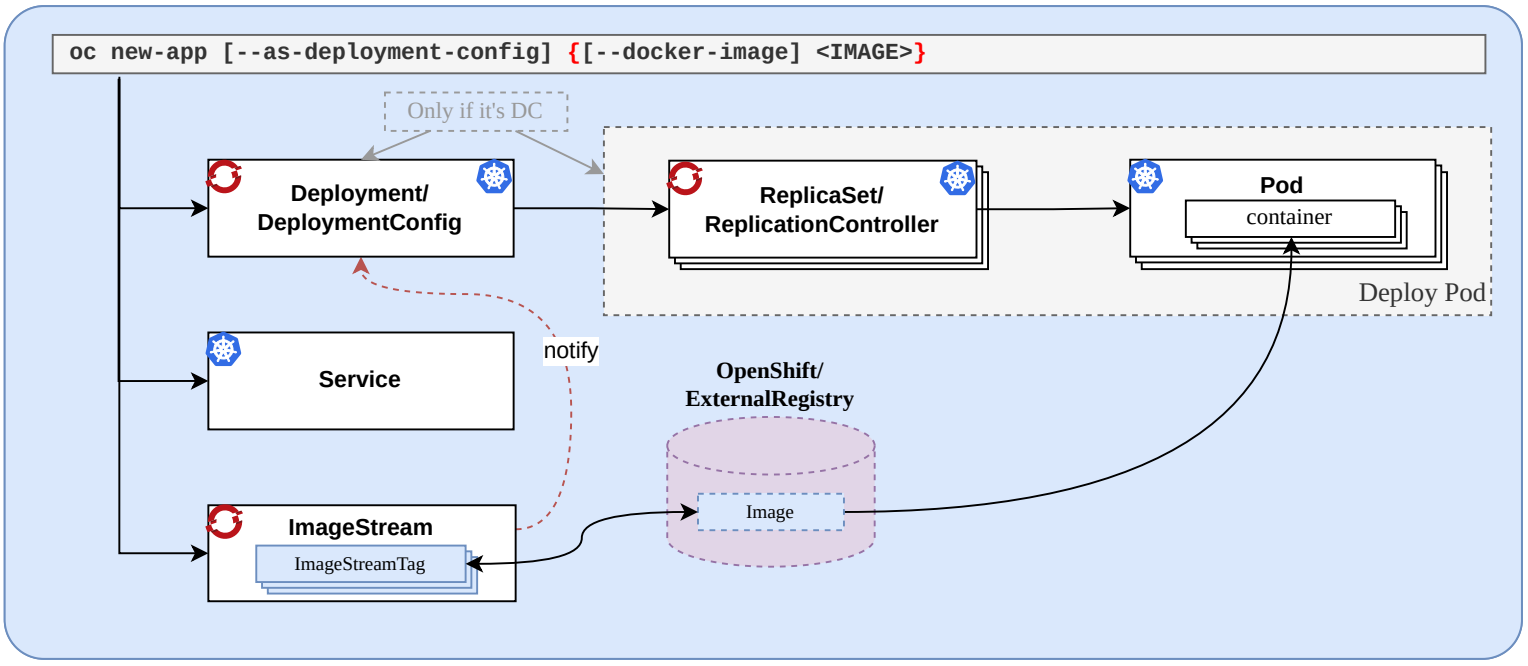
3. Using yaml/json file

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

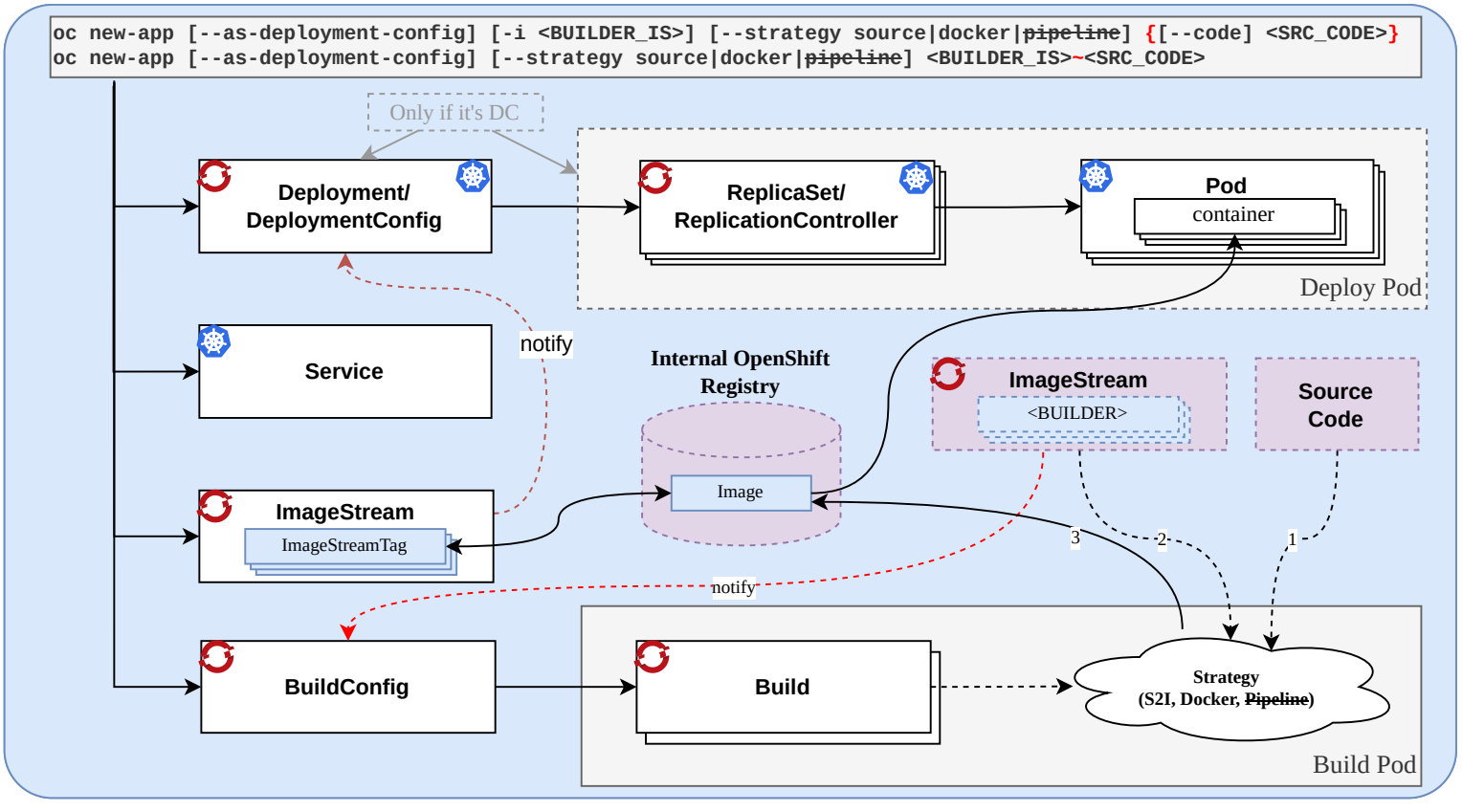
4. Using template

```
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.

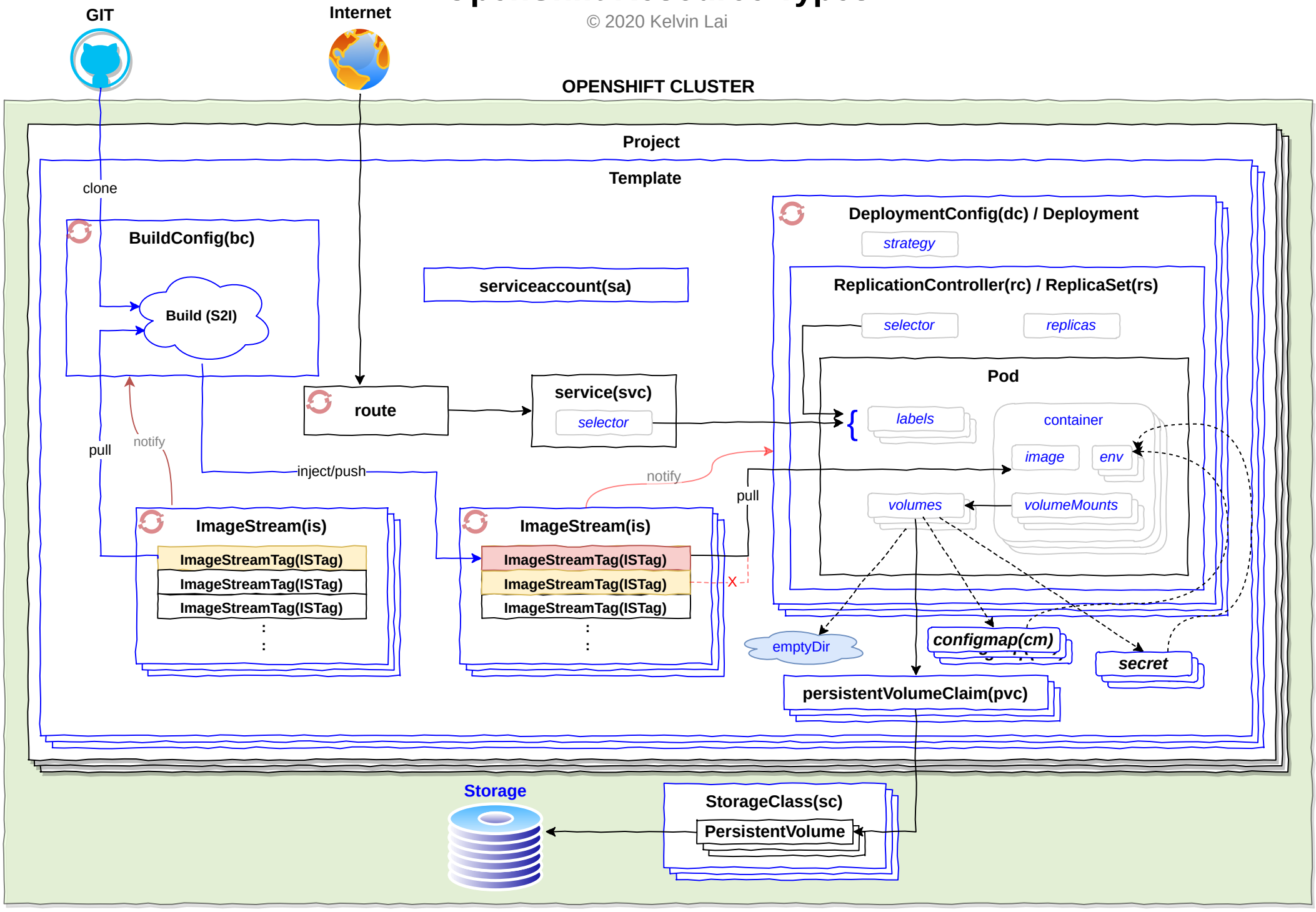
oc expose <DC/DEPLOYMENT/RC/RS/POD> <RESOURCE_NAME>

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>

OpenShift Resource Types



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

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