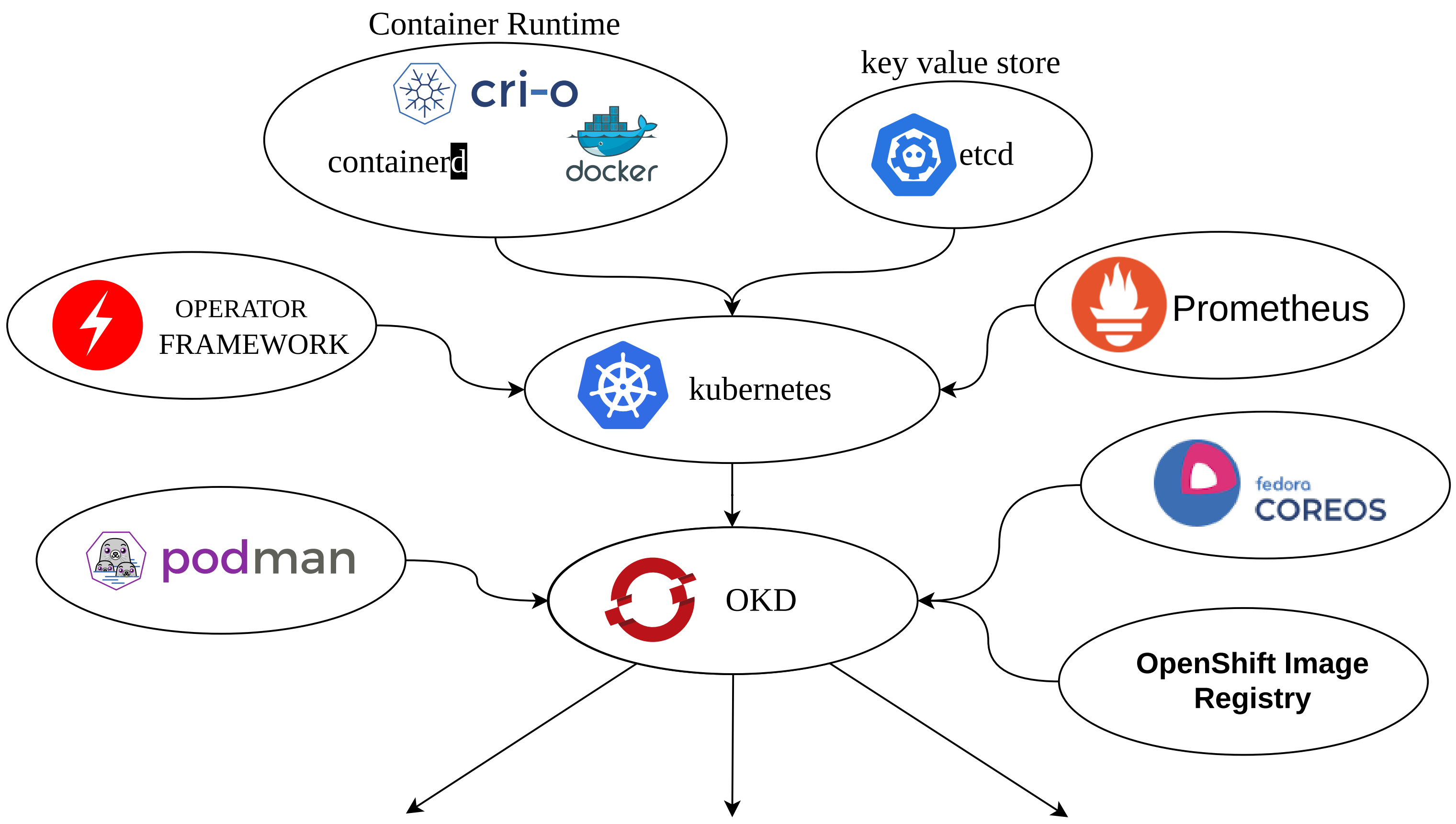


Red Hat OpenShift Container Platform

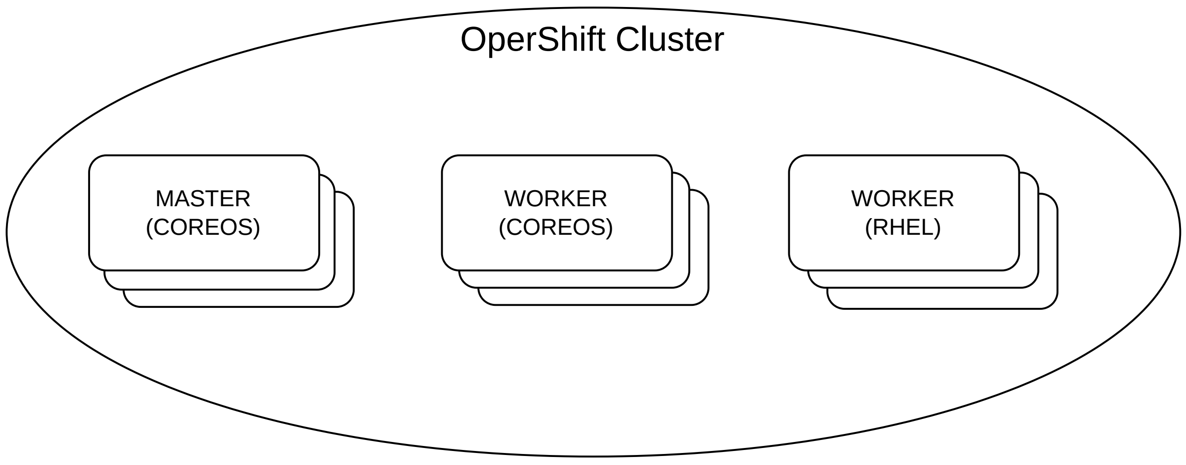
- Public/private DC.
- Bare metal and multiple cloud and virtualization providers.
- Full control by customer.

Red Hat OpenShift Dedicated

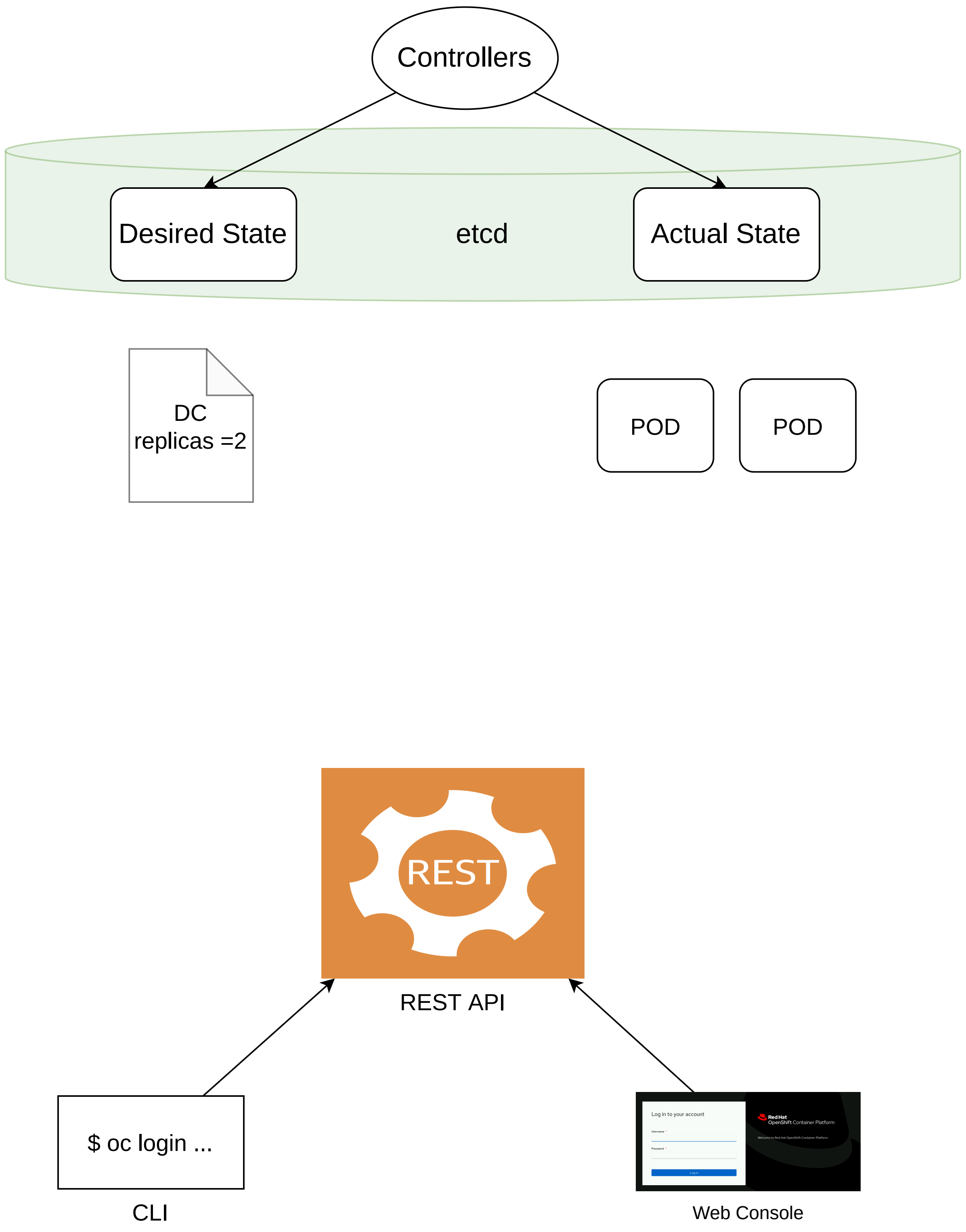
- Managed cluster in public cloud.
- RH manages the cluster.
- Customer manages updates and add-on services.

Red Hat OpenShift Online

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

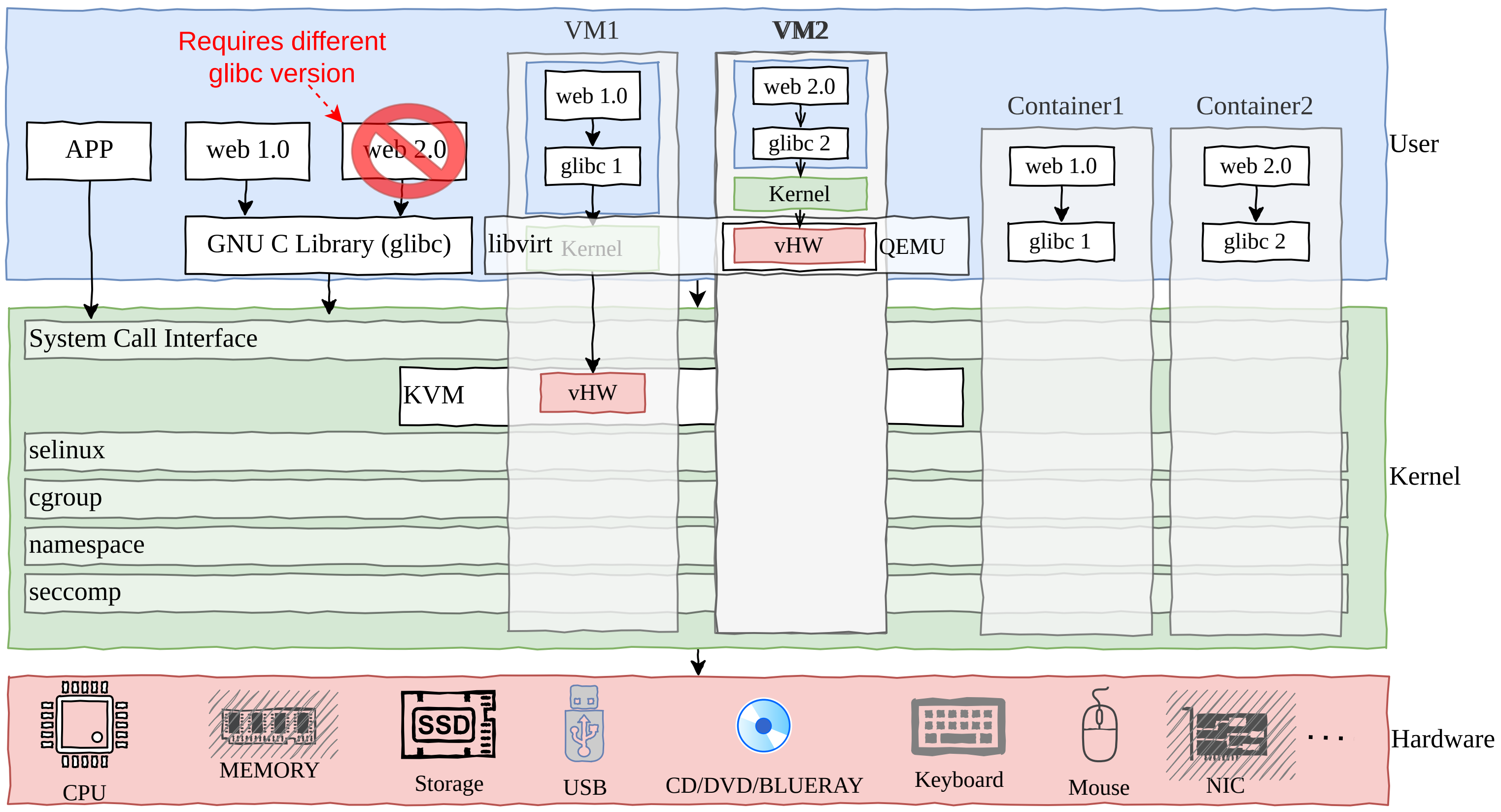


Kubernetes Declarative Architecture

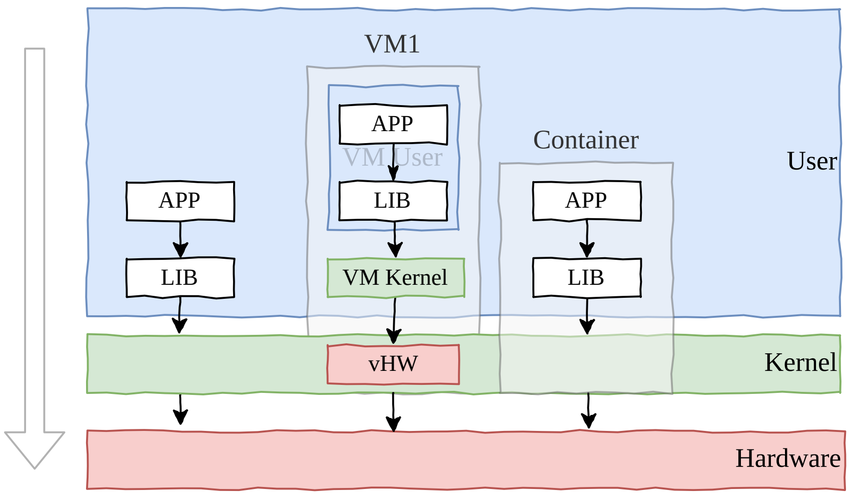


Welcome to Red Hat OpenShift Container Platform.

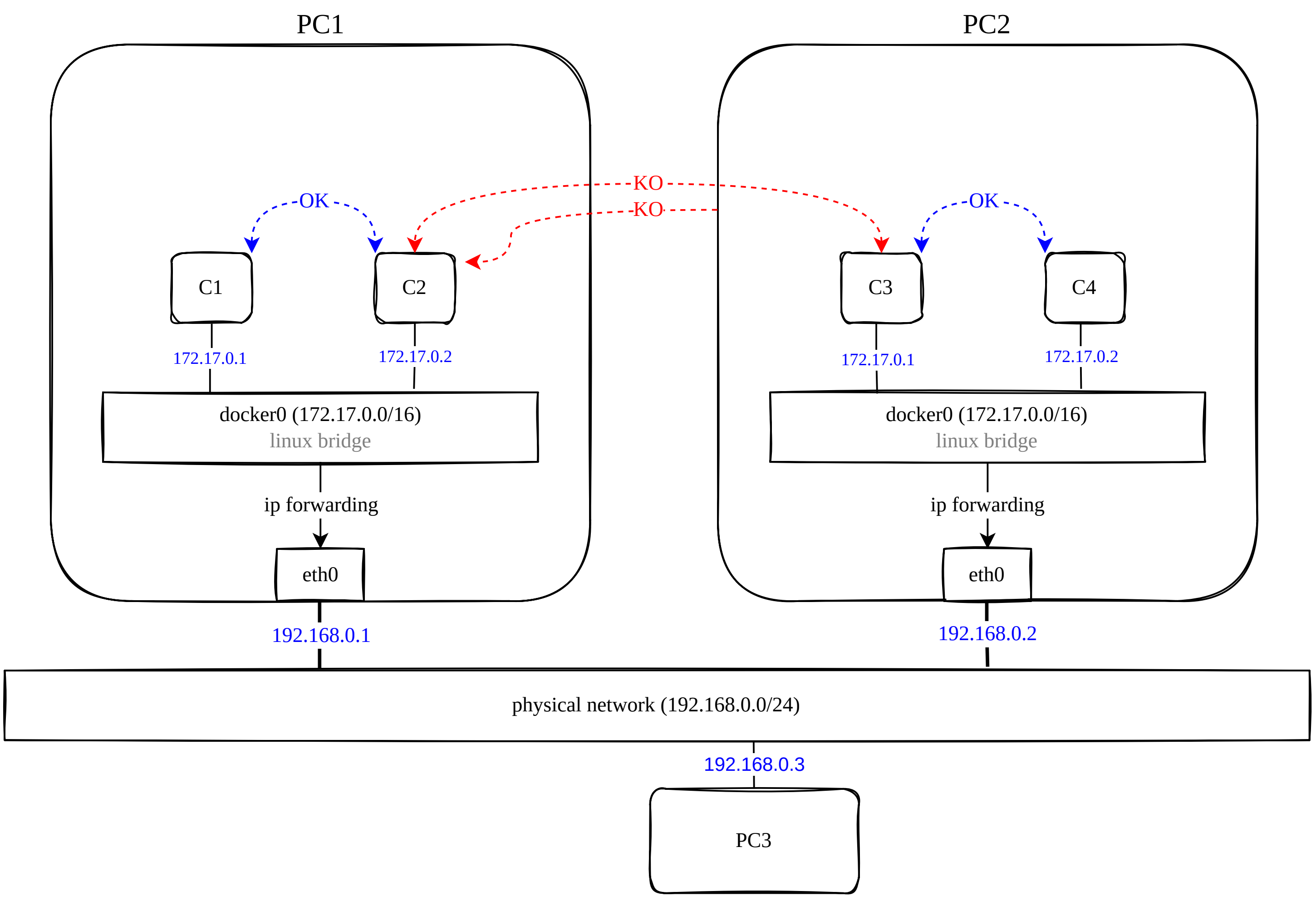
VM vs Container

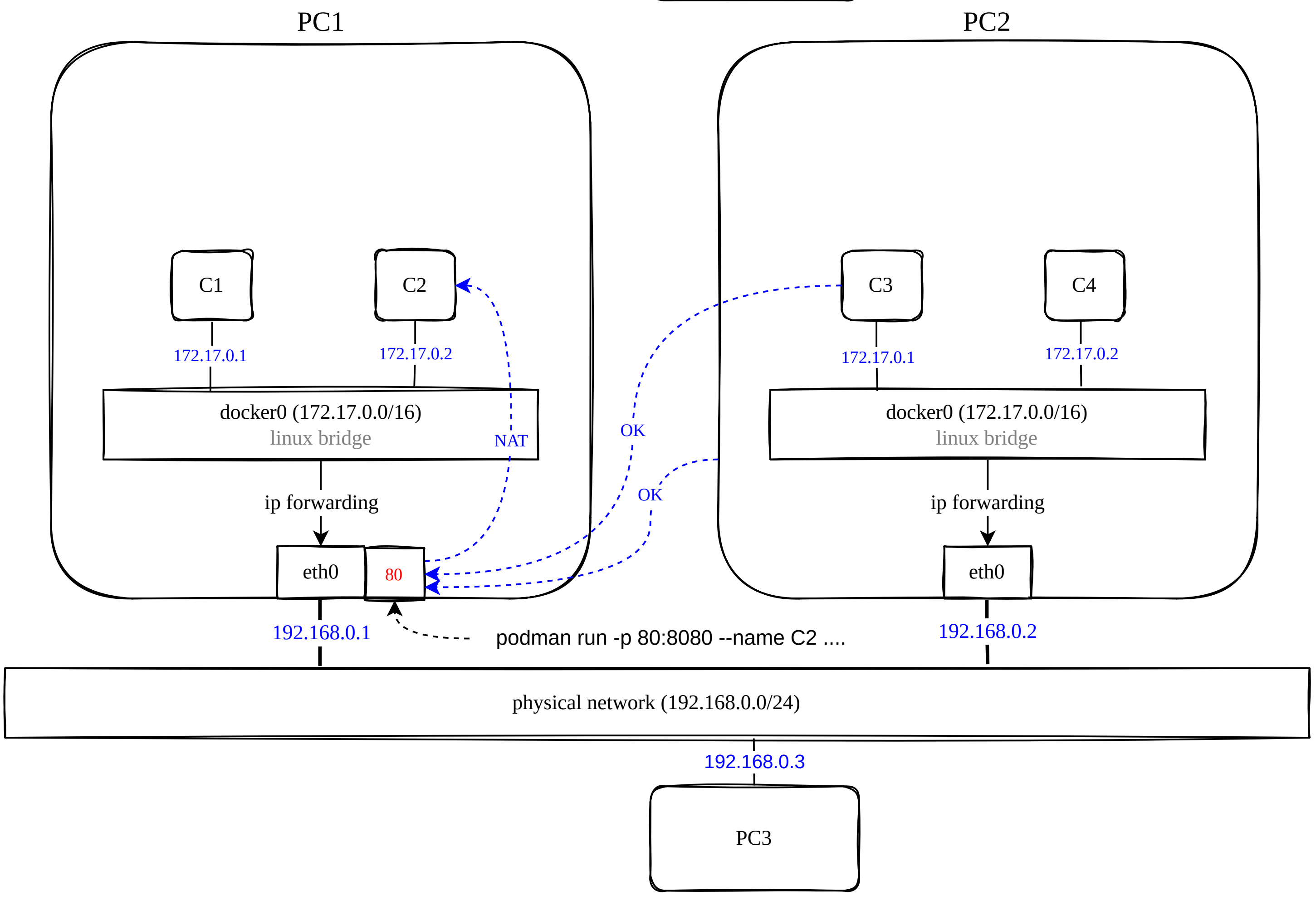


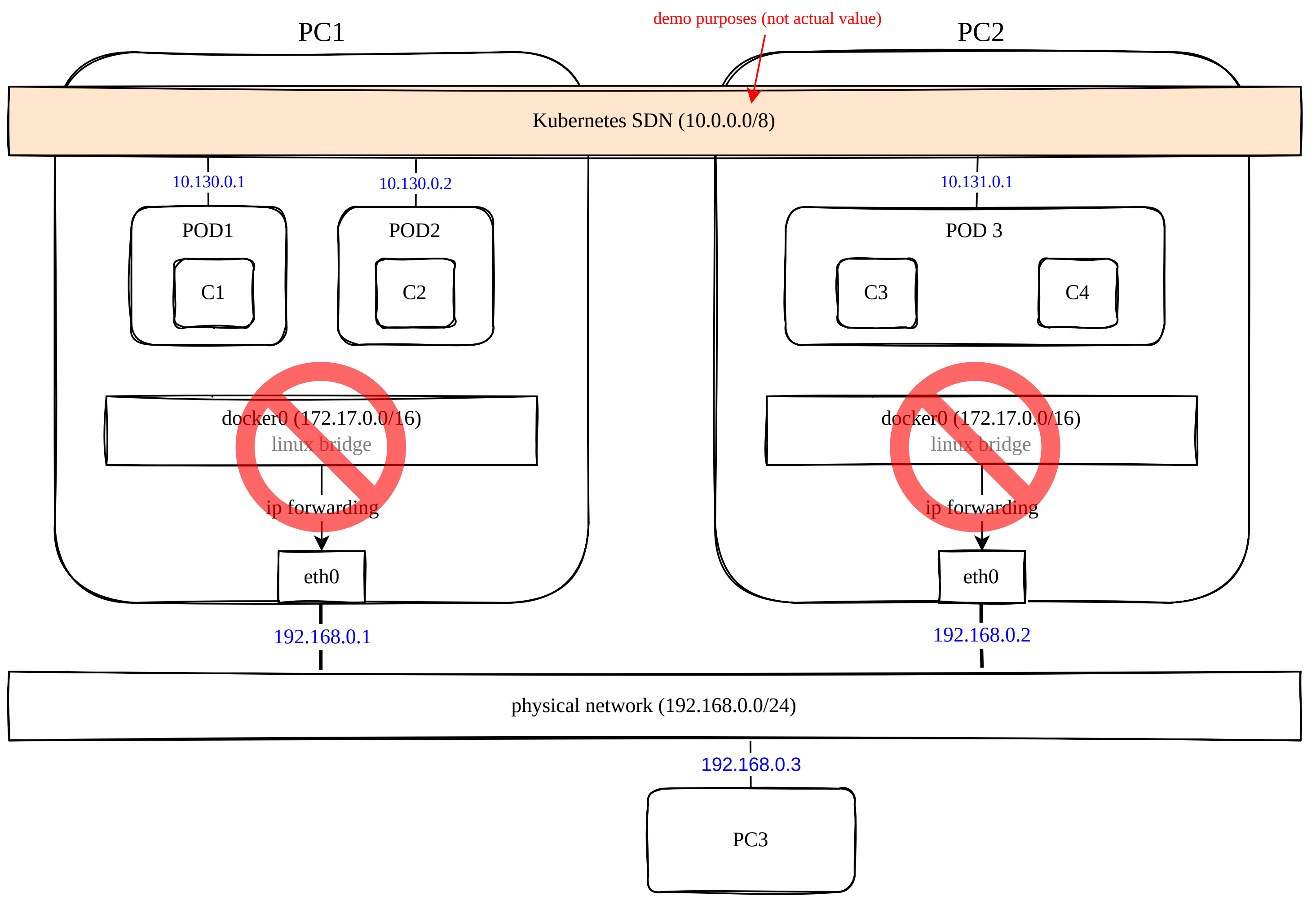
Ref: https://www.redhat.com/en/blog/all-you-need-know-about-kvm-userspace
https://www.packetcoders.io/what-is-the-difference-between-qemu-and-kvm/



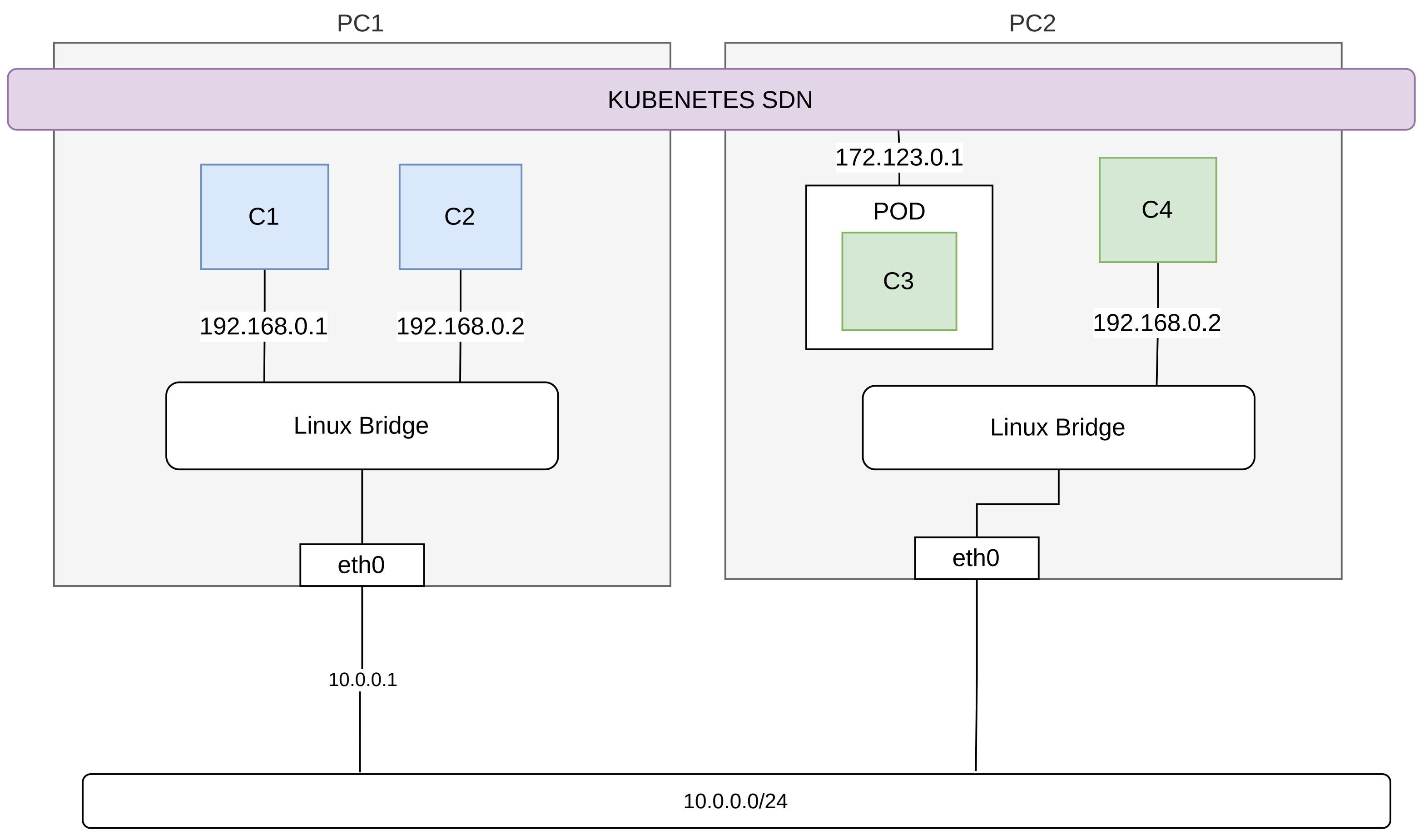
Basic Network - Container vs Kubernetes





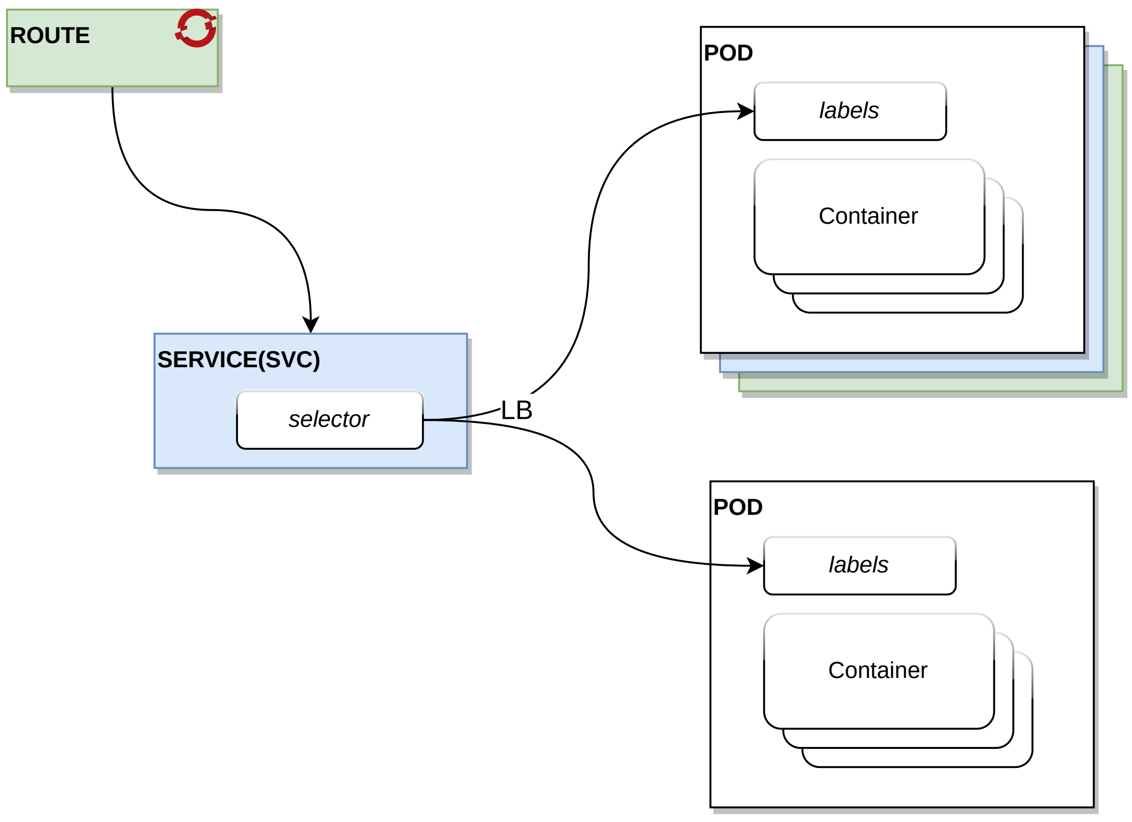


PC1
PC2
demo purposes (not actual value)
Kubernetes SDN (10.0.0.0/8)
10.130.0.1
10.130.0.2
10.131.0.1
POD1
POD2
POD 3
C1
C2
C3
C4
docker0 (172.17.0.0/16)
linux bridge
ip forwarding
eth0
docker0 (172.17.0.0/16)
linux bridge
ip forwarding
eth0
192.168.0.1
192.168.0.2
physical network (192.168.0.0/24)
192.168.0.3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



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

Route, Service and Pod Relationship



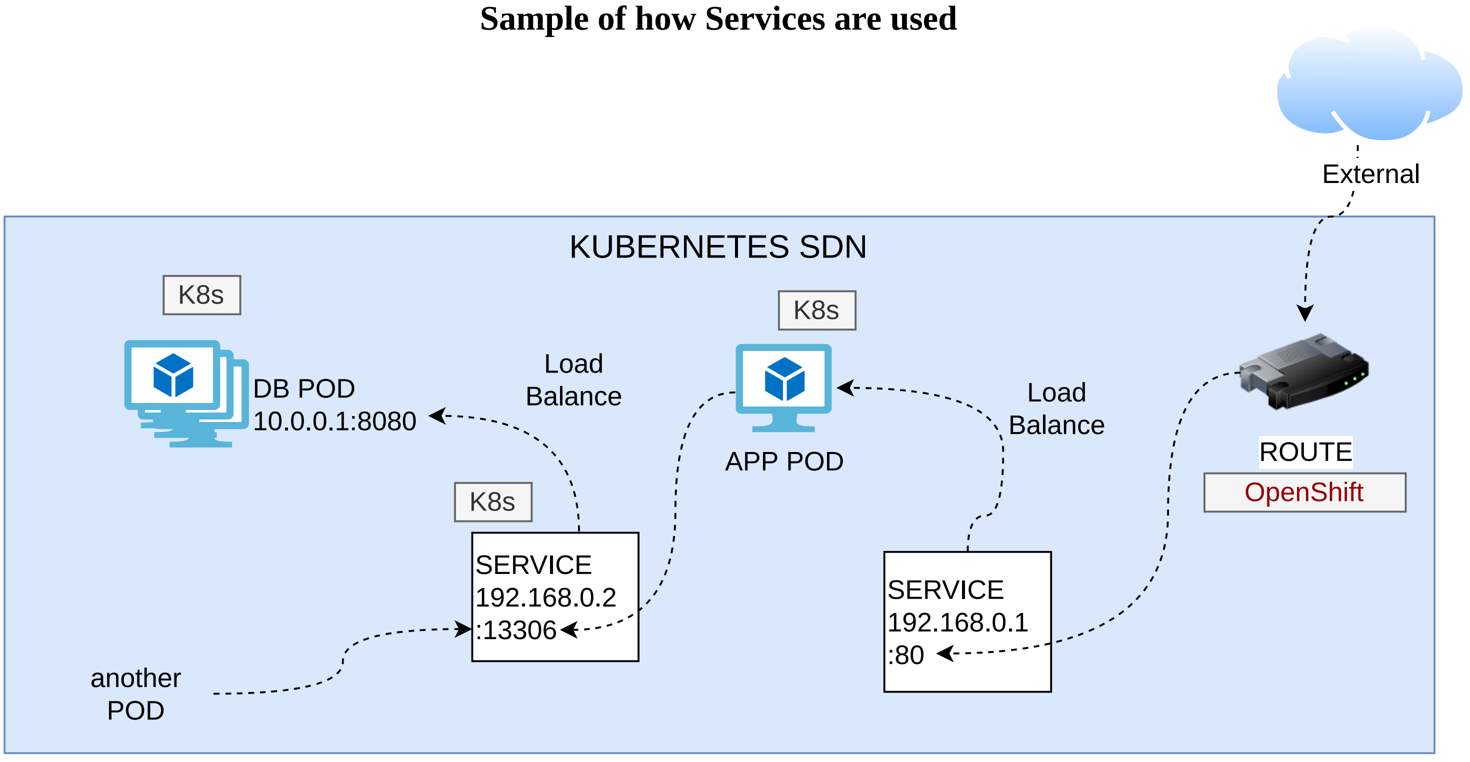
<u>POD</u>
A pod contains one or more containers.

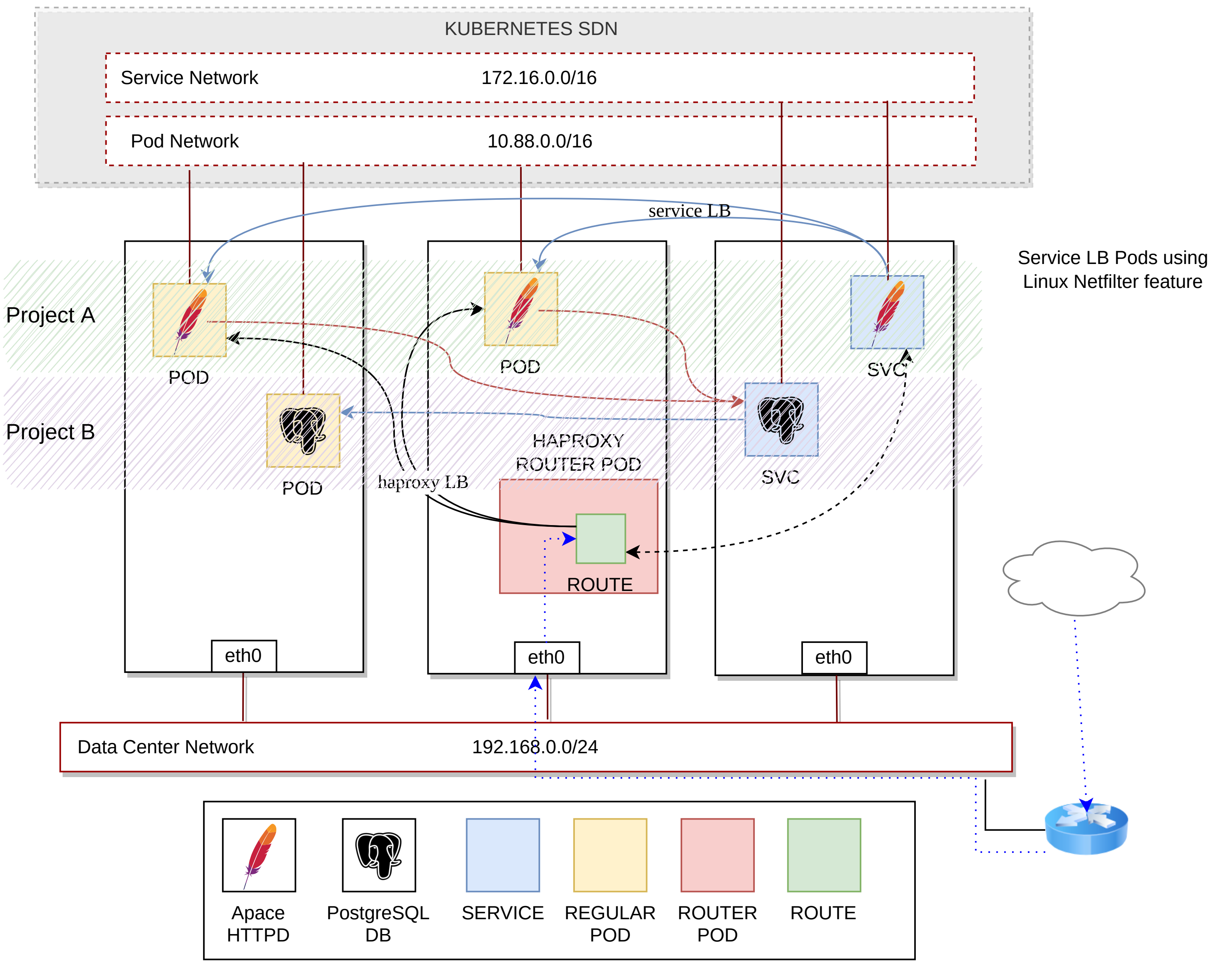
<u>SERVICE</u>
A service references the pod(s) by using the label selector.
The service load balances the connections between all the pods.

<u>ROUTE</u>
A route exposes the service to the external world.

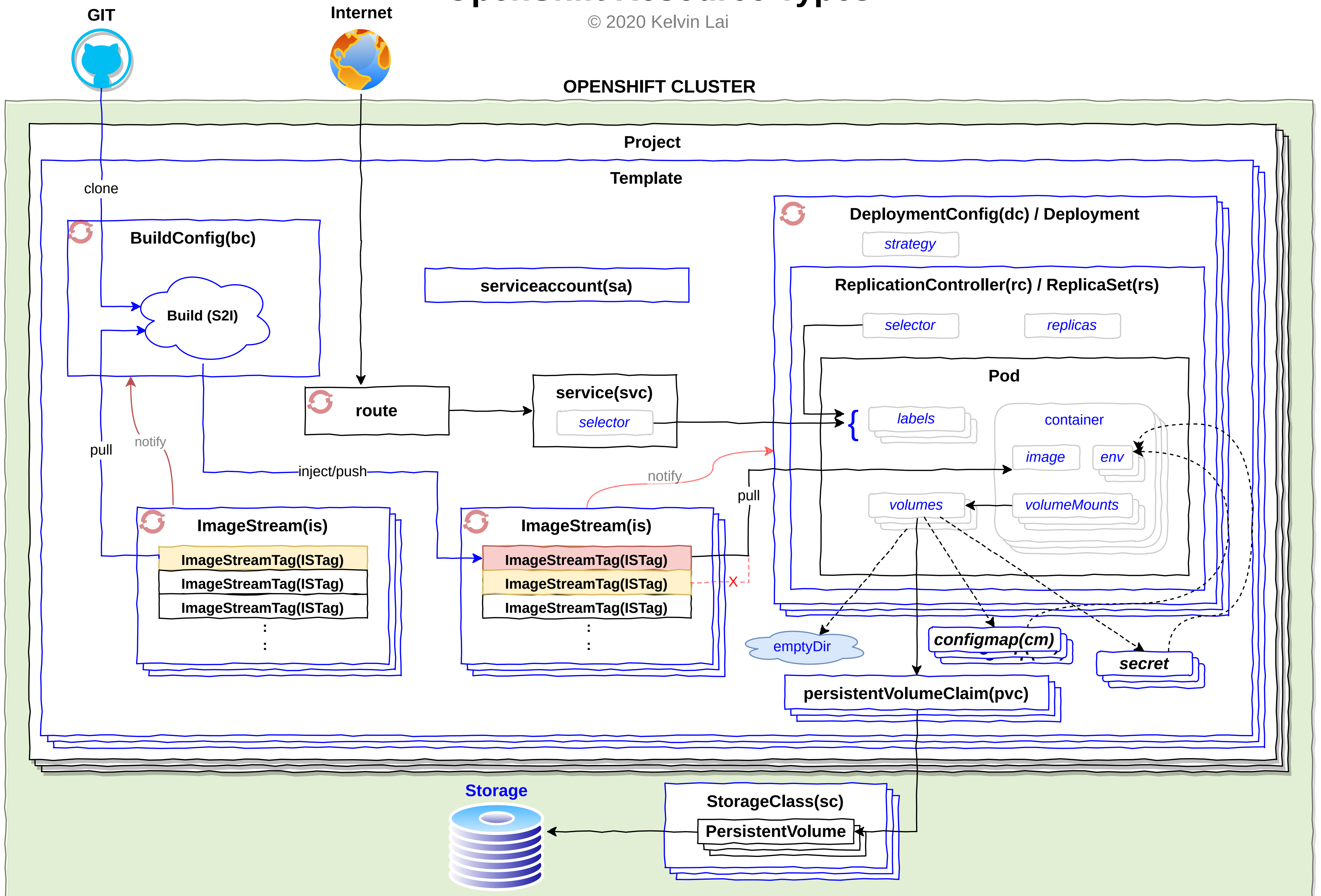
Warning: A service "can" refer to different pods, if the pods have the same label.

Sample of how Services are used



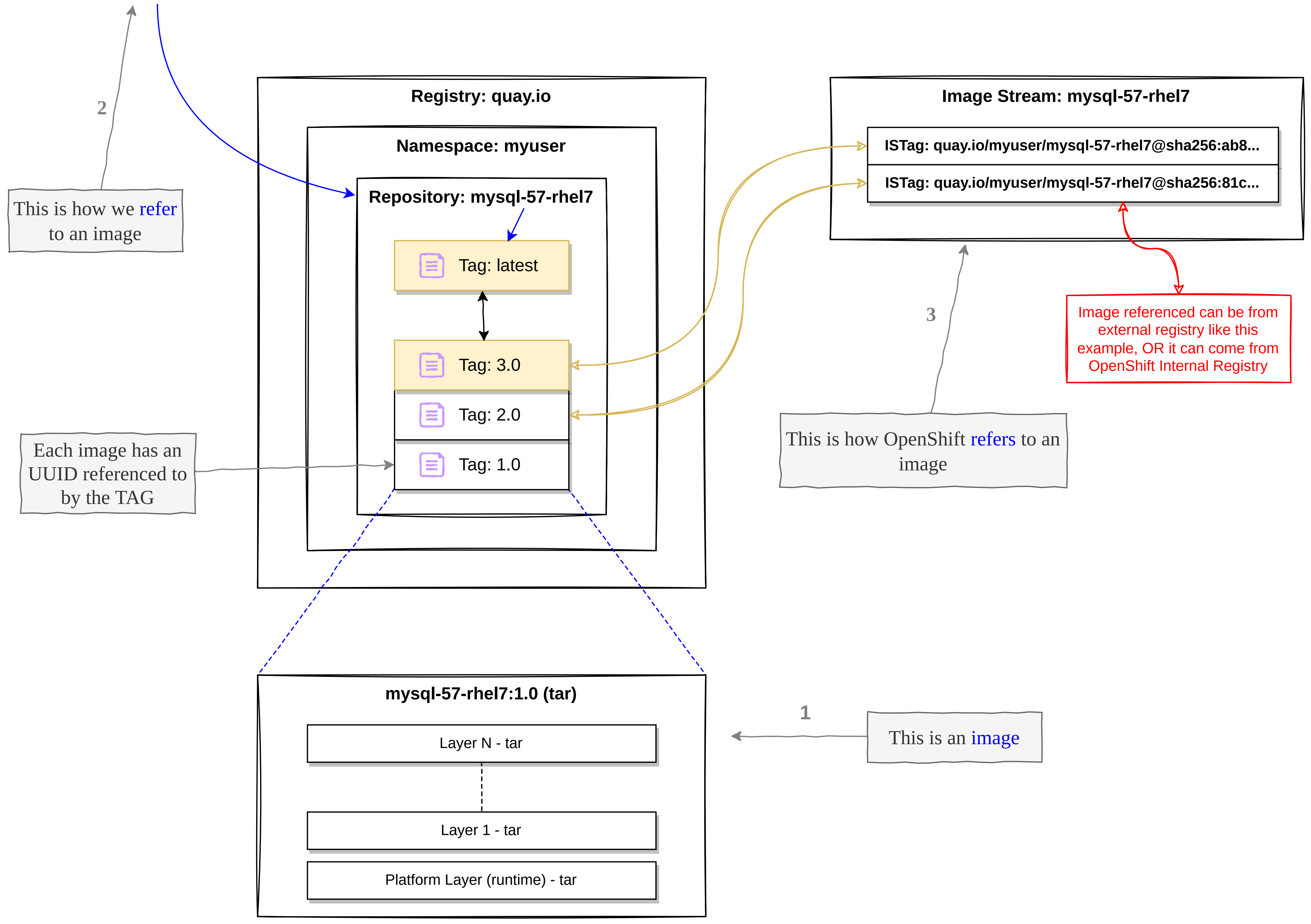


OpenShift Resource Types



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

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



2
This is how we refer to an image
Registry: quay.io
Namespace: myuser
Repository: mysql-57-rhel7
Tag: latest
Tag: 3.0
Tag: 2.0
Tag: 1.0
Each image has an UUID referenced to by the TAG
Image Stream: mysql-57-rhel7
ISTag: quay.io/myuser/mysql-57-rhel7@sha256:ab8...
ISTag: quay.io/myuser/mysql-57-rhel7@sha256:81c...
3
Image referenced can be from external registry like this example, OR it can come from OpenShift Internal Registry
This is how OpenShift refers to an image
mysql-57-rhel7:1.0 (tar)
Layer N - tar
Layer 1 - tar
Platform Layer (runtime) - tar
1
This is an image

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>
```

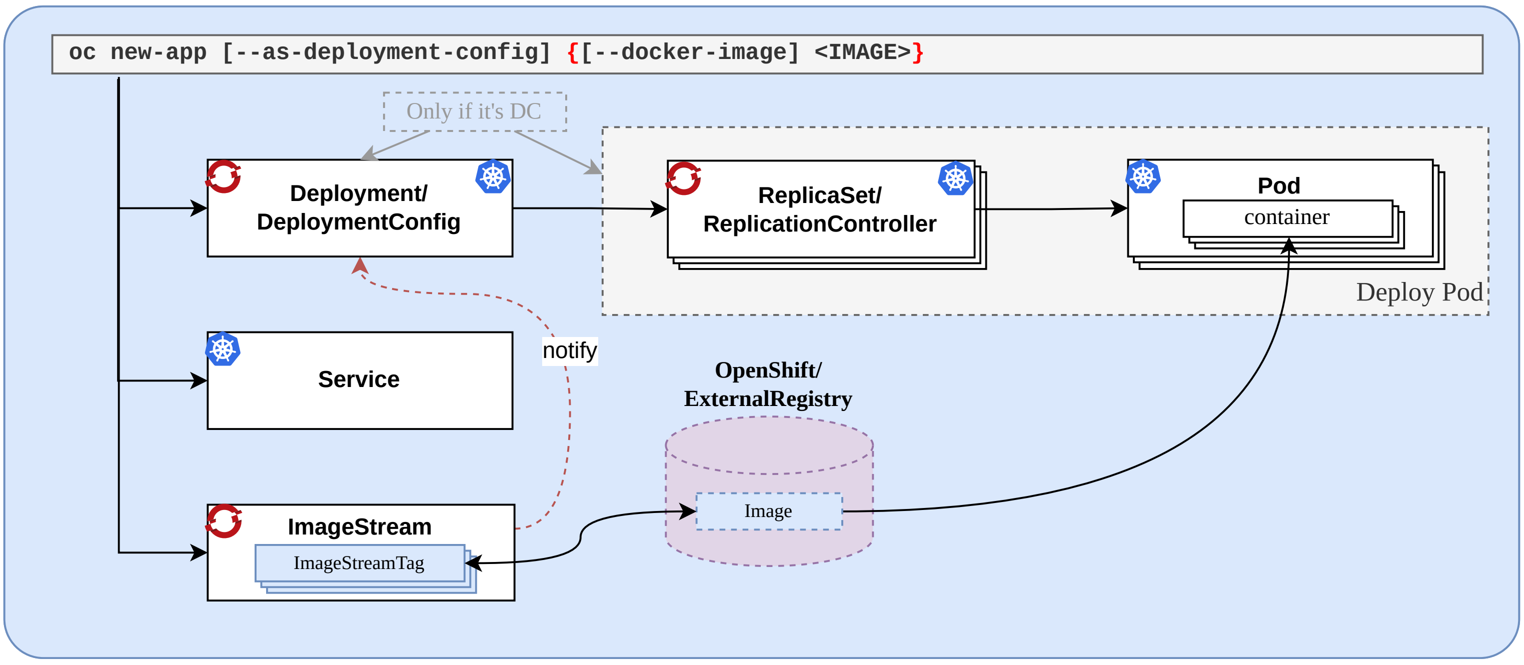
3. Using yaml/json file

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

4. Using template

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

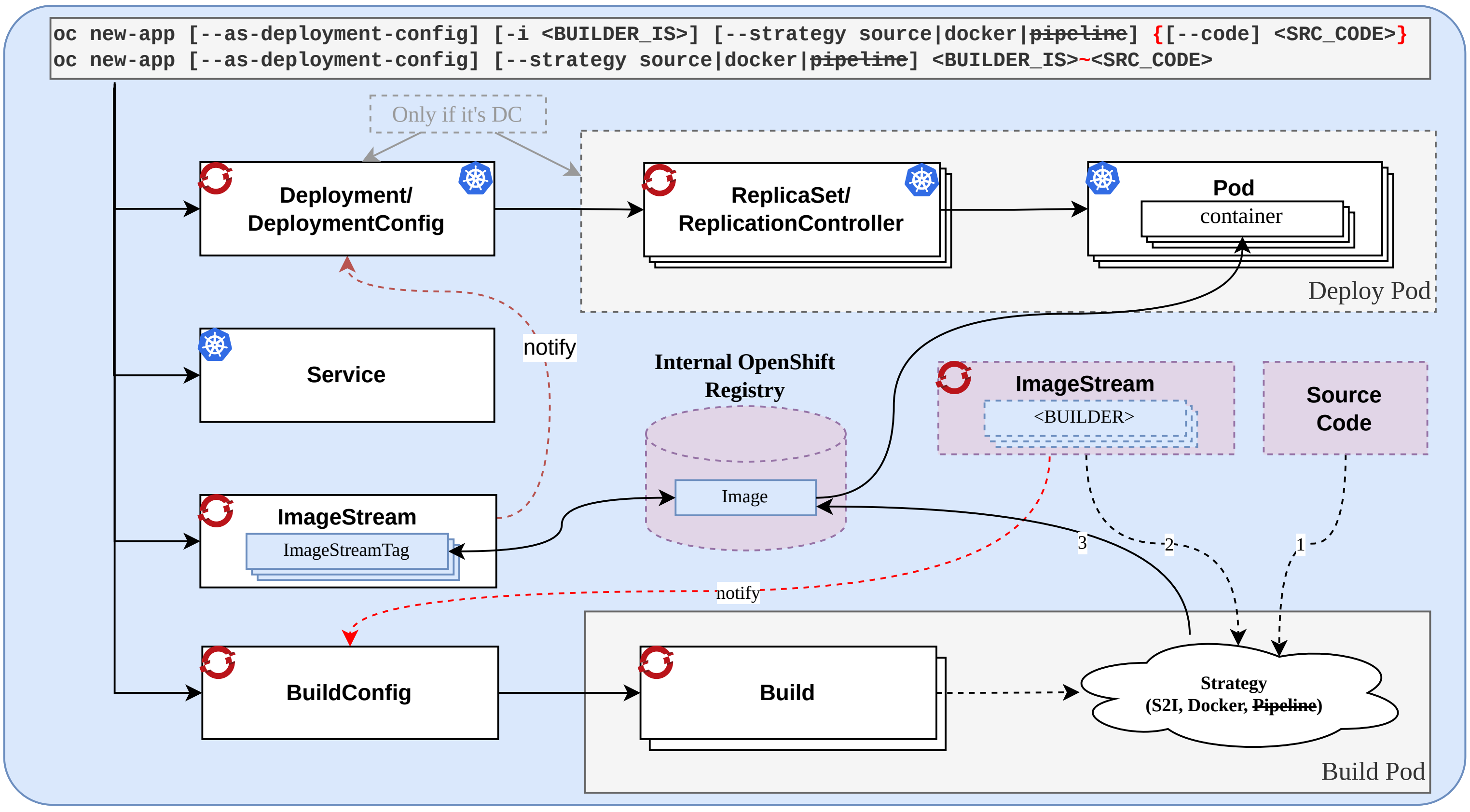
1. Use Existing Image



```
oc new-app [--as-deployment-config] {[--docker-image] <IMAGE>}
```

Only if it's DC
Deployment/ DeploymentConfig
ReplicaSet/ ReplicationController
Pod
container
Deploy Pod
Service
notify
OpenShift/ ExternalRegistry
Image
ImageStream
ImageStreamTag

2. Managed Life Cycle



```
oc new-app [--as-deployment-config] [-i <BUILDER_IS>] [--strategy source|docker|~~pipeline~~] {[--code] <SRC_CODE>}
oc new-app [--as-deployment-config] [--strategy source|docker|~~pipeline~~] <BUILDER_IS>~<SRC_CODE>
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.

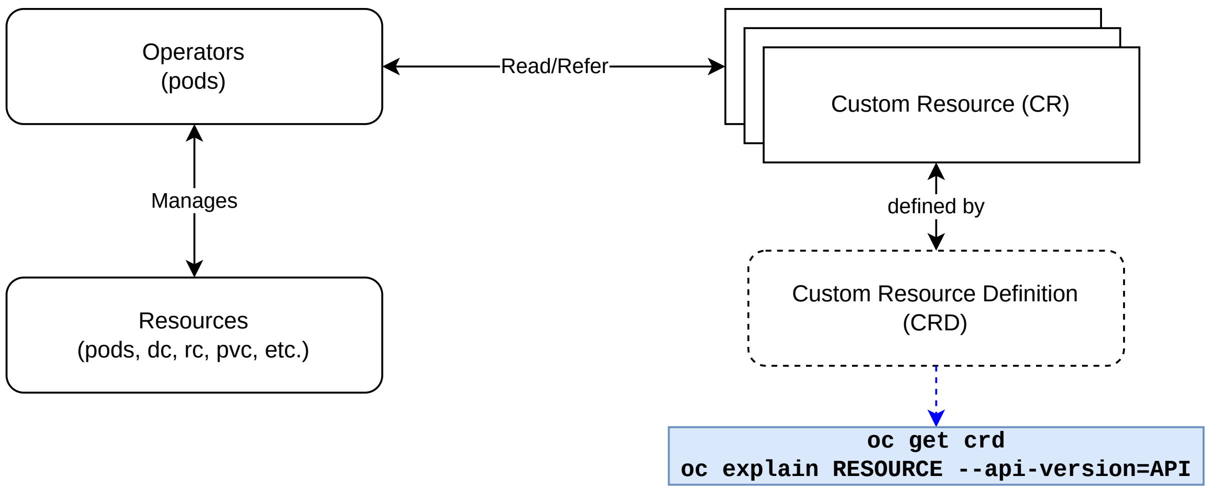
SERVICE(SVC)

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

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

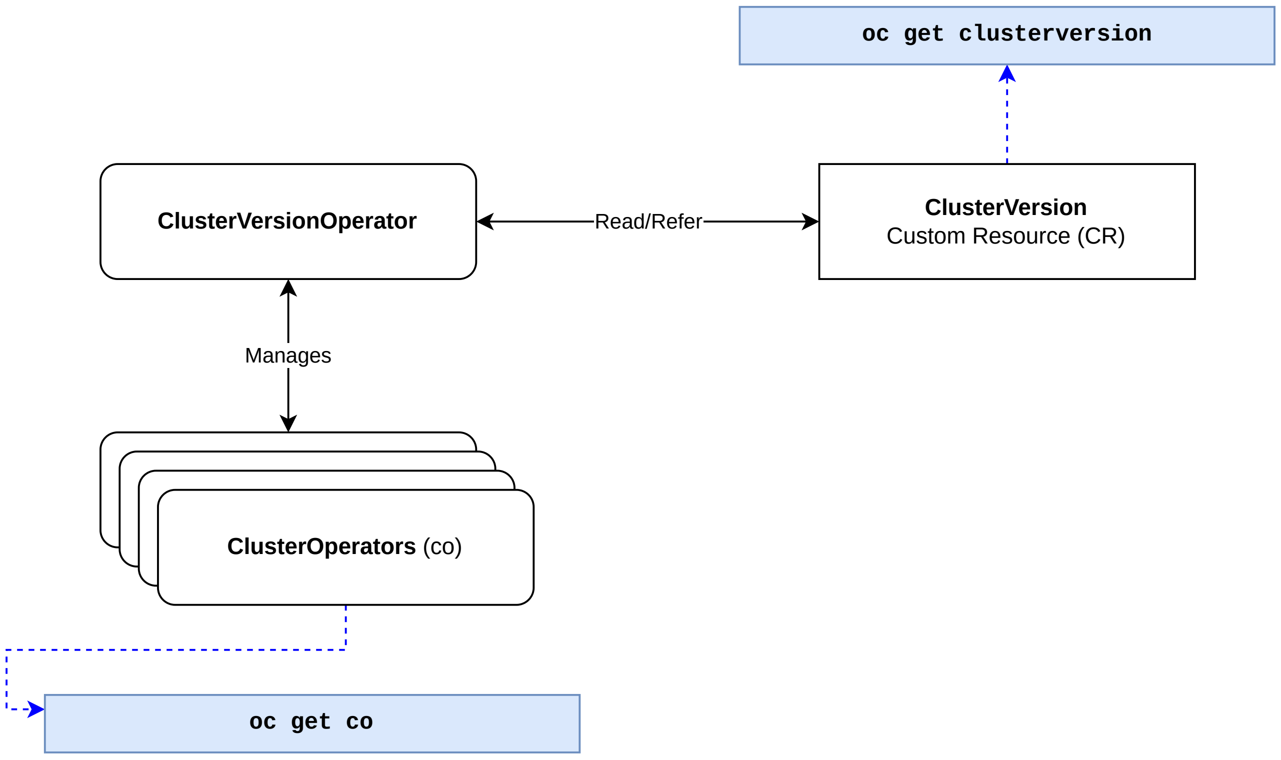
ROUTE

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

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

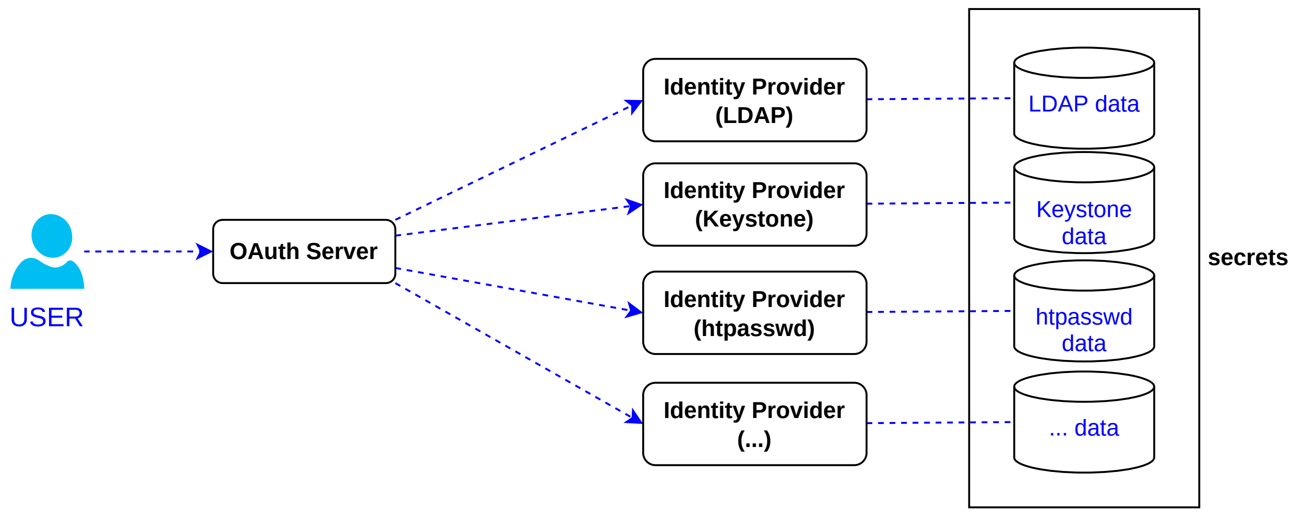
Operators

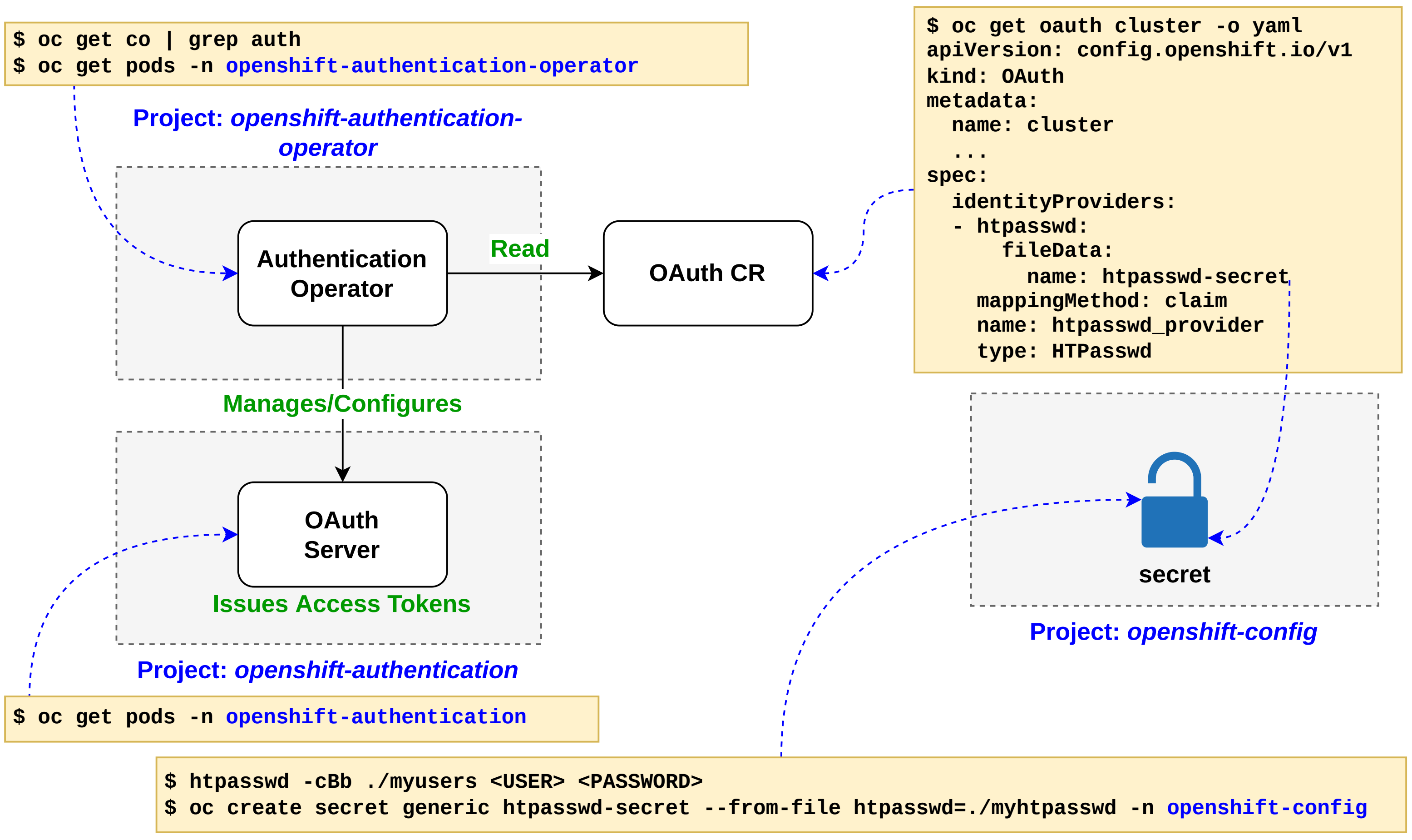


```
oc get crd
oc explain RESOURCE --api-version=API
```



```
oc get clusterversion
oc get co
```



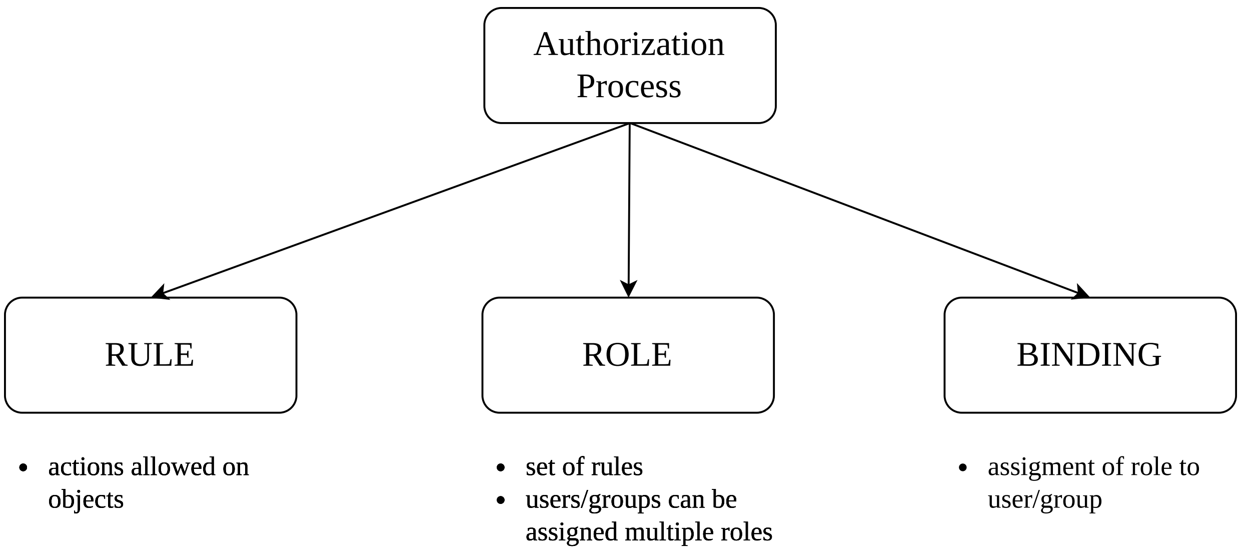

Project: openshift-authentication-operator
Authentication Operator
Read
OAuth CR
Manages/Configures
OAuth Server
Issues Access Tokens
Project: openshift-authentication
secret
Project: openshift-config

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

```
$ 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 pods -n openshift-authentication
```

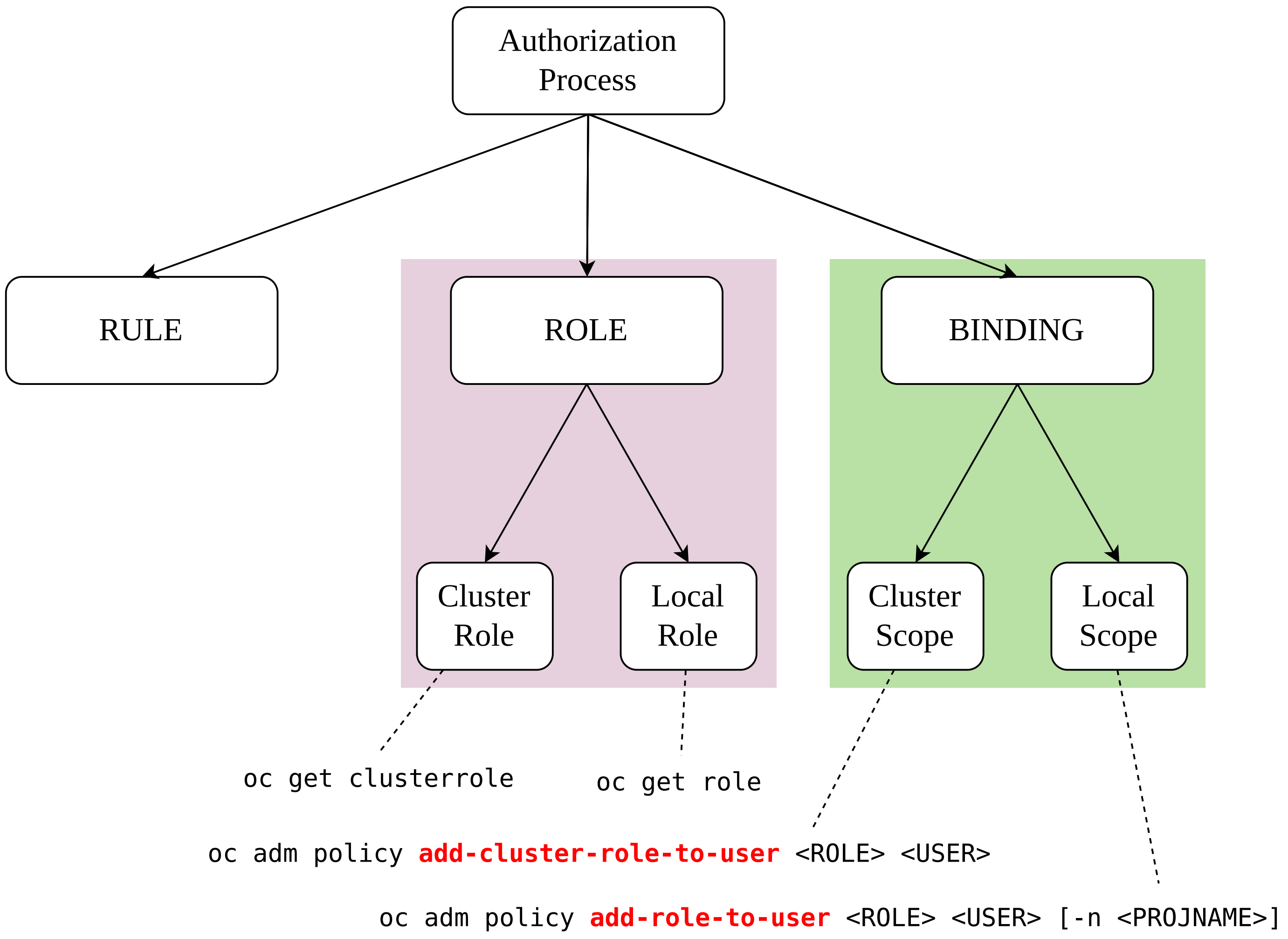
```
$ htpasswd -cBb ./myusers <USER> <PASSWORD>
$ oc create secret generic htpasswd-secret --from-file htpasswd=./myhtpasswd -n openshift-config
```



- actions allowed on objects

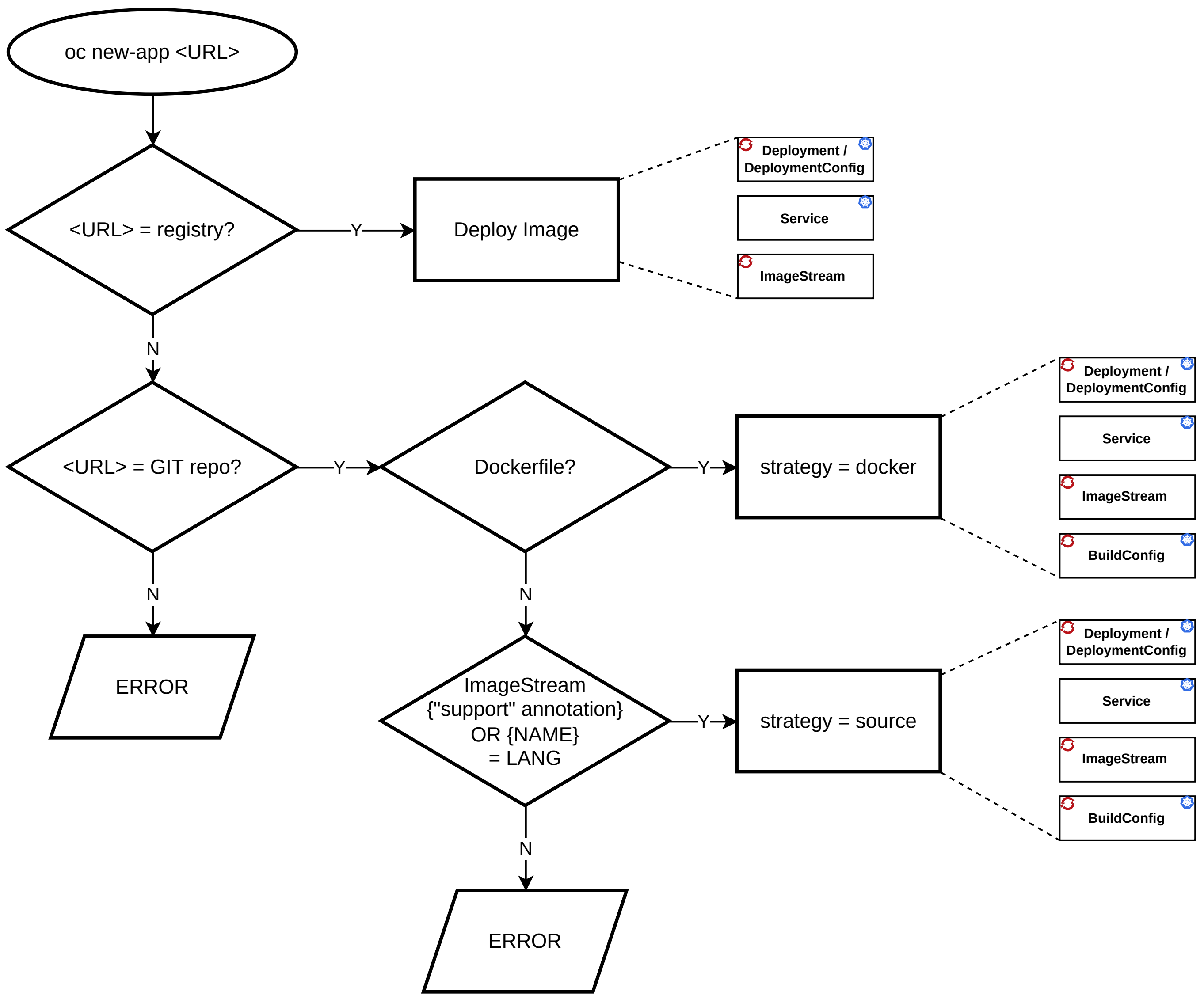
- set of rules
- users/groups can be assigned multiple roles

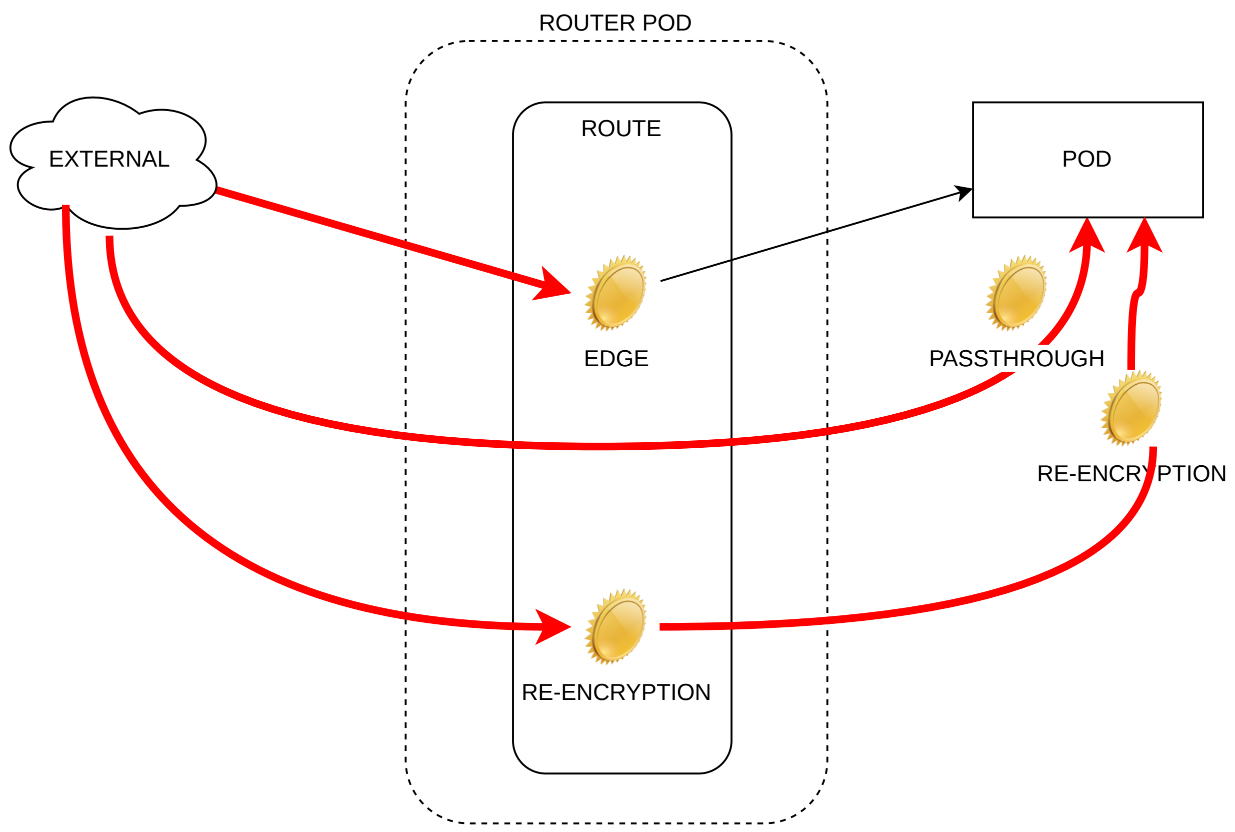
- assigment of role to user/group

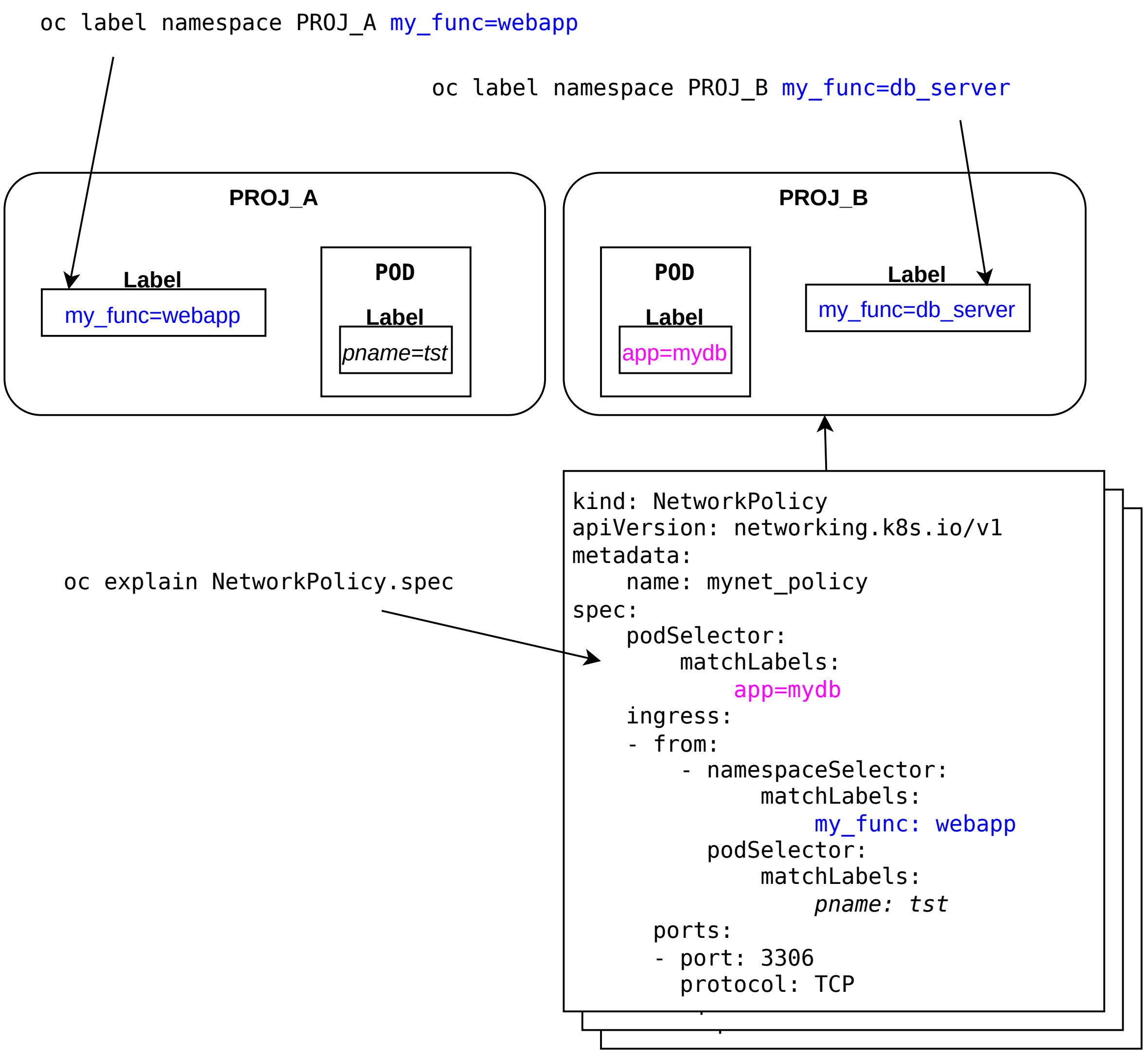


```
oc get clusterrole
oc get role
oc adm policy add-cluster-role-to-user <ROLE> <USER>
oc adm policy add-role-to-user <ROLE> <USER> [-n <PROJNAME>]
```

oc new-app flowchart







```
oc label namespace PROJ_A my_func=webapp
```

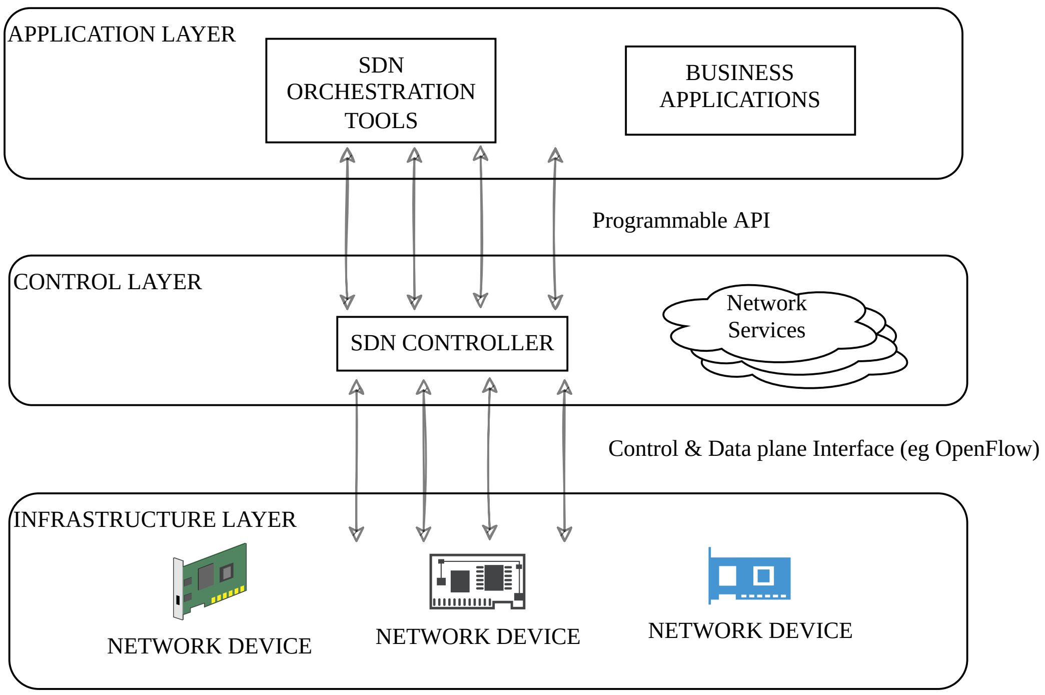
```
oc label namespace PROJ_B my_func=db_server
```

```
oc explain NetworkPolicy.spec
```

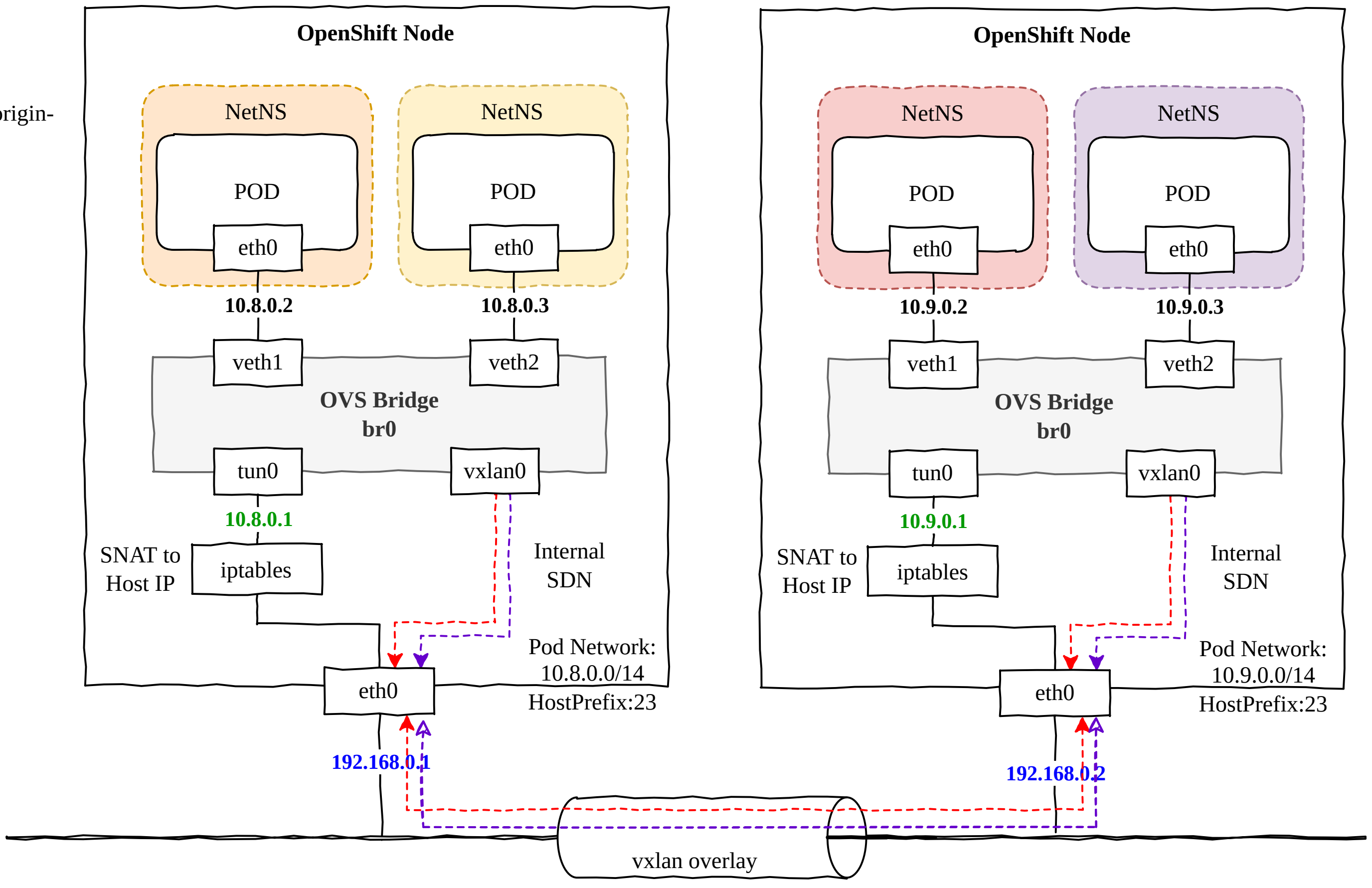
```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
    name: mynet_policy
spec:
    podSelector:
        matchLabels:
            app=mydb
    ingress:
    - from:
        - namespaceSelector:
              matchLabels:
                  my_func: webapp
          podSelector:
              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

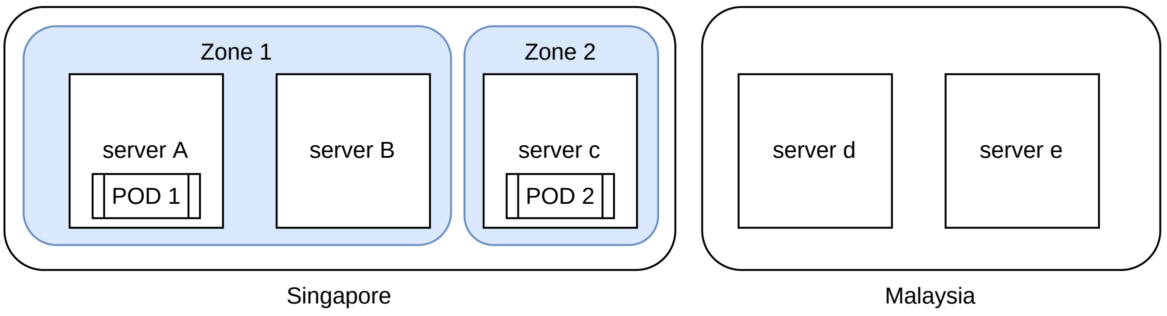
POD Scheduling

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.



server a = 0 + 0
server b = 1 + 0
server c = 1 + 1
~~server d~~
~~server e~~

```
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