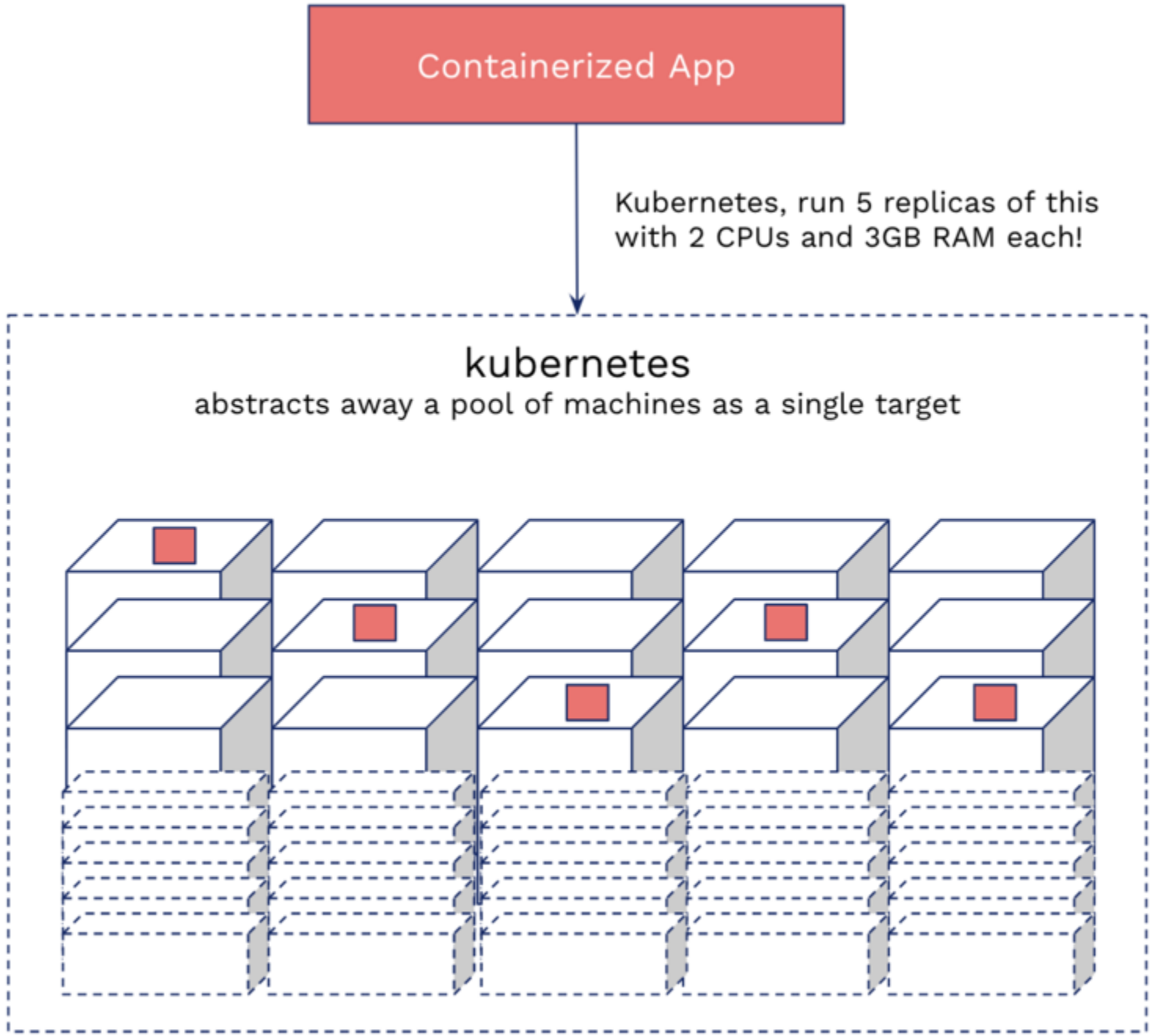
Kubernetes Internals & Security

Hyunsang Choi

Kubernetes?

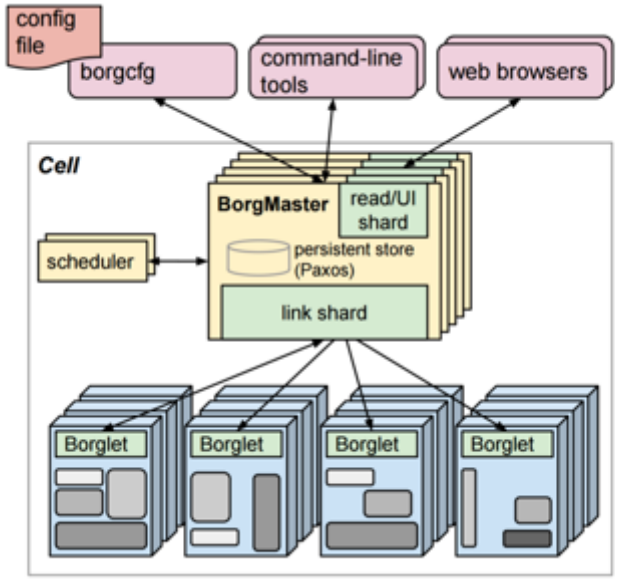

Solved all your problems. You're welcome.
I NEED TO KNOW WHY MOVING OUR APP TO THE CLOUD DIDN'T AUTOMATICALLY SOLVE ALL OUR PROBLEMS.
YOU WOULDN'T LET ME RE-ARCHITECT THE APP TO BE CLOUD-NATIVE.
JUST PUT IT IN CONTAINERS.
YOU CAN'T SOLVE A PROBLEM JUST BY SAYING TECHY THINGS.
KUBERNETES.
Dilbert.com @ScottAdamsSays
11-08-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel

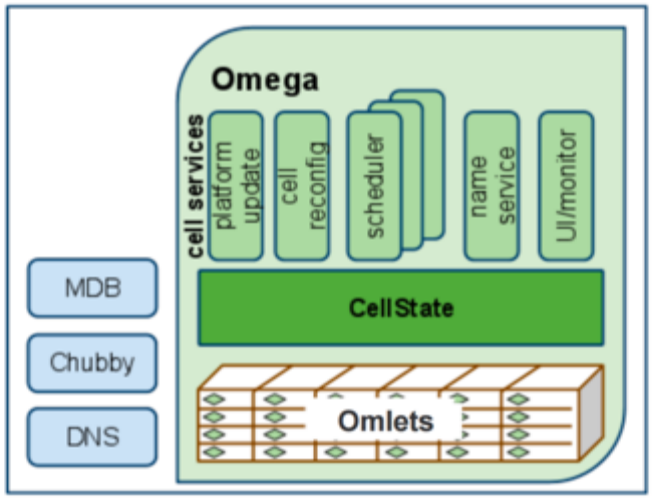


Layman's representation of Kubernetes

Borg & Omega

- Google’s cluster orchestration projects
  - 10/15 years R&D (secret)
  - Borg: Large scale cluster management (written in C++)
  - Omega: Improve performance



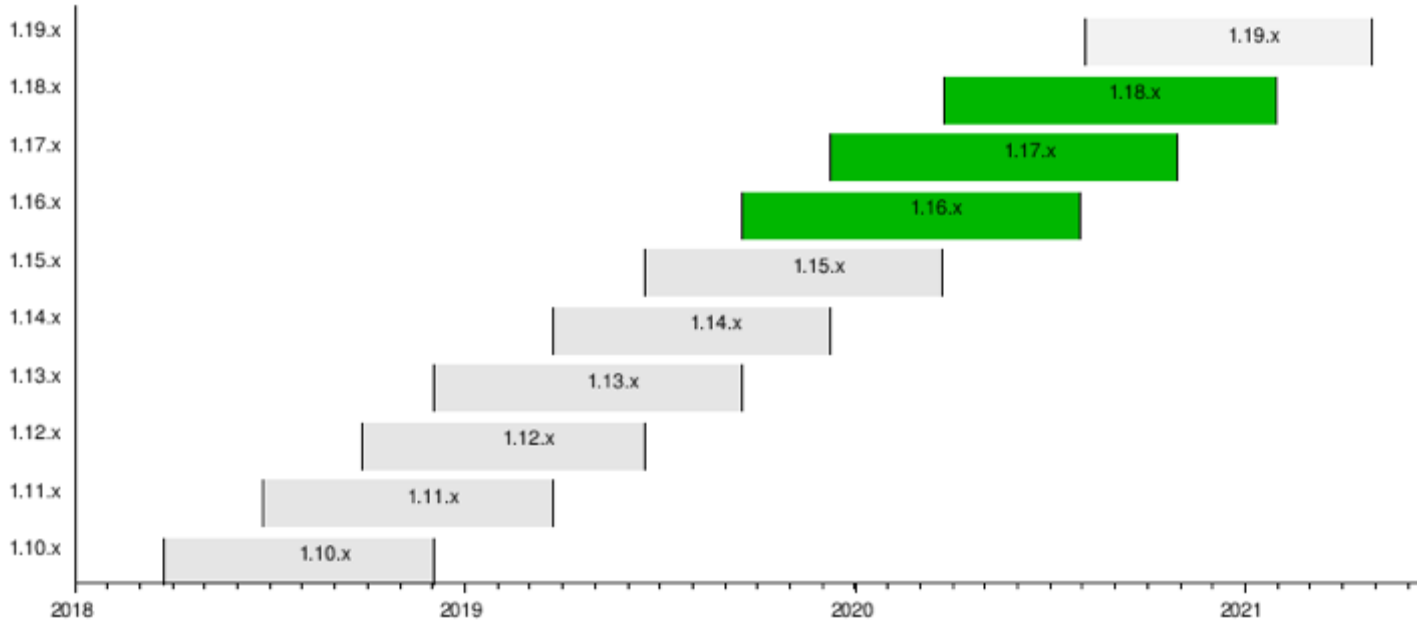


Borg vs Kubernetes

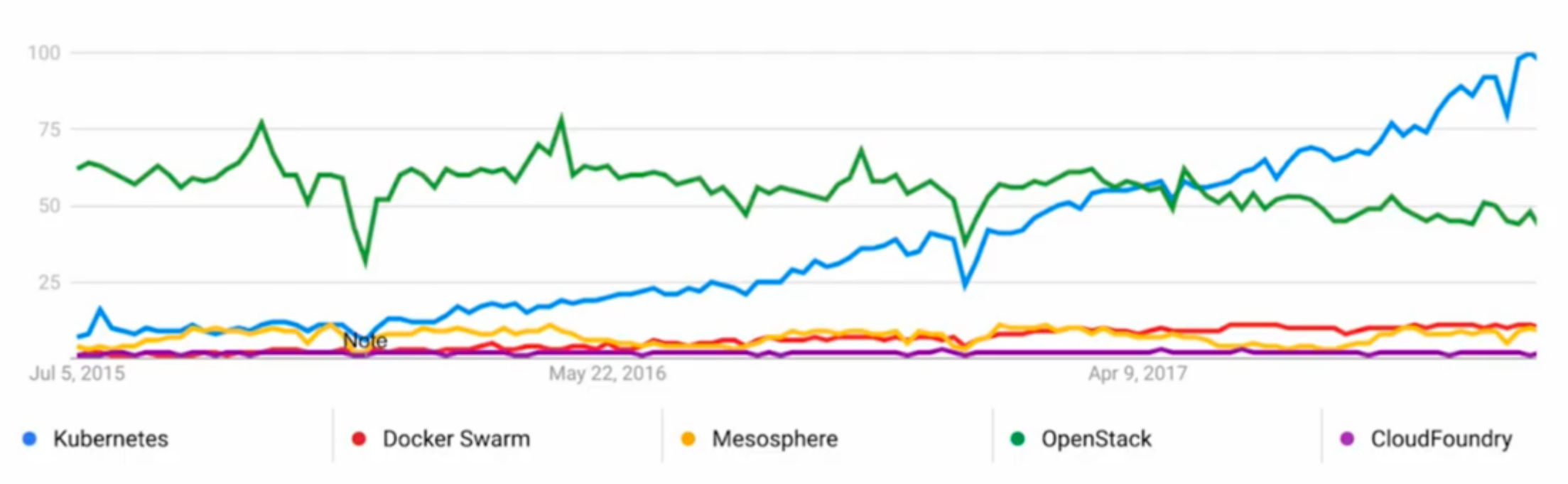
- Borg groups work by ‘job’; Kubernetes adds ‘labels’ for greater flexibility
- Borg uses an IP-per-machine design; Kubernetes uses a network-per-machine and IP-per-Pod design to allow late-binding of ports (letting developers choose ports, not the infrastructure)
- Kubernetes APIs are simpler than Borg's API
- Borg uses LMCTFY as its container technology; Kubernetes allows the use of Docker, rkt, etc
- Borg is not open source

Kubernetes

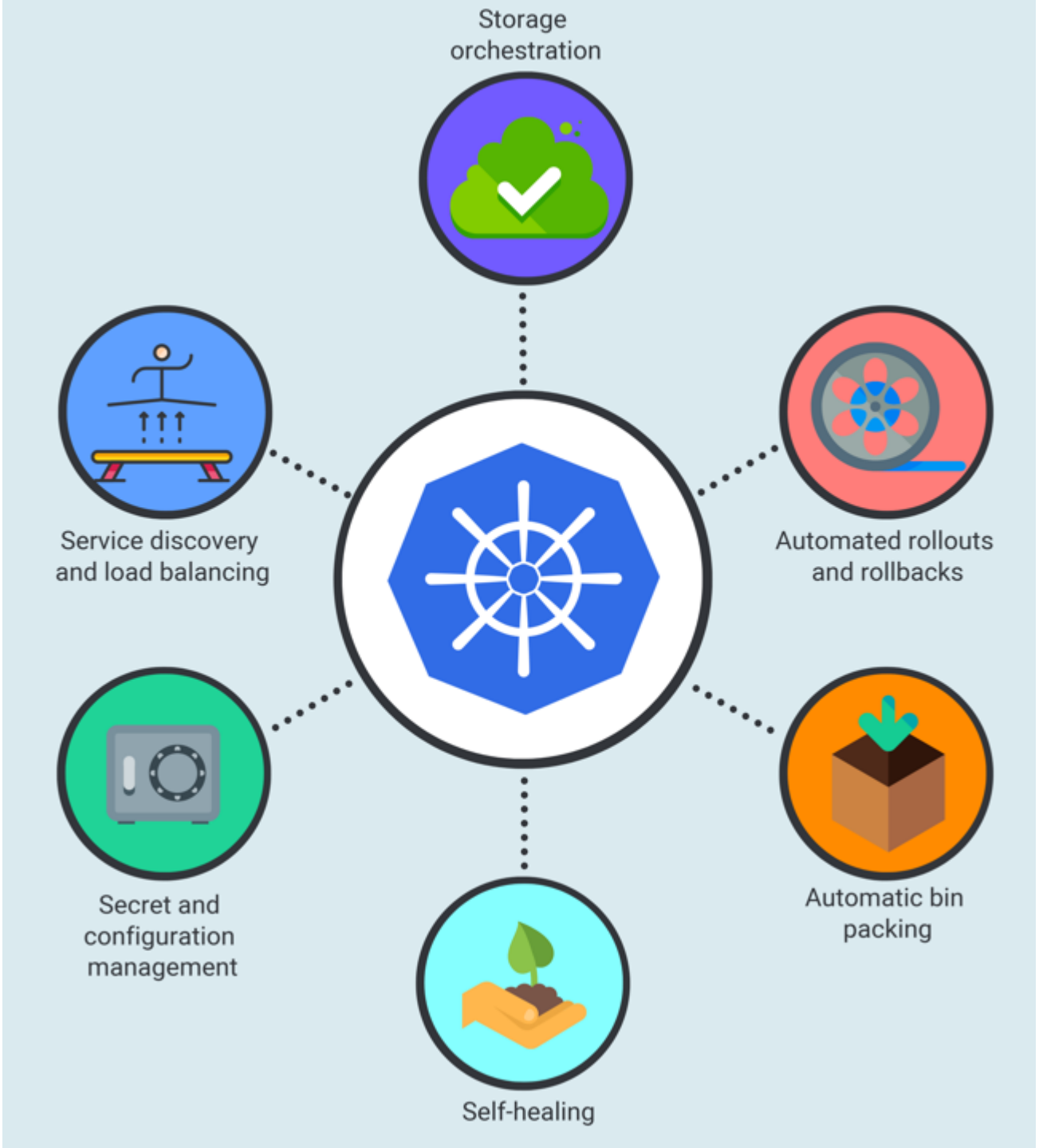
- Open-source container-orchestration system for automating application deployment, scaling, and management
- Greek (κυβερνήτης): helmsman
- Heavily influenced by Borg project
- V1.0: 2015.07.21
- Implemented in Go
- Apache License 2.0
- Maintained by CNCF
- Stylized as k8s



De-Facto Standard

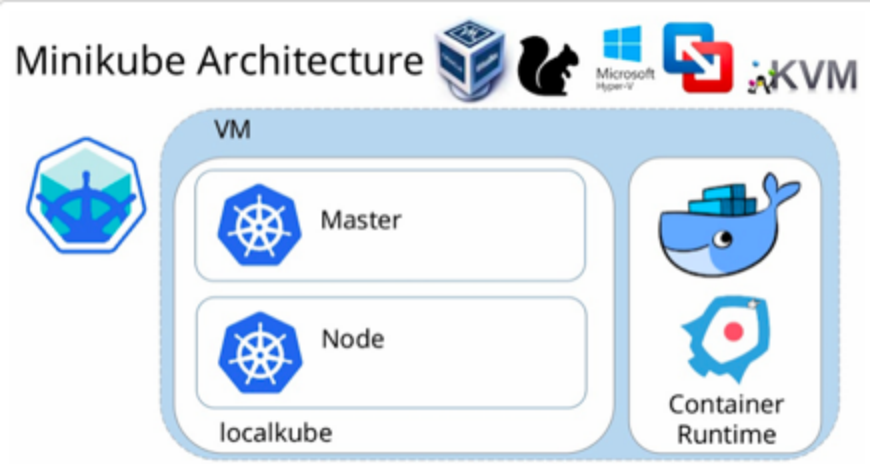


k8s Features

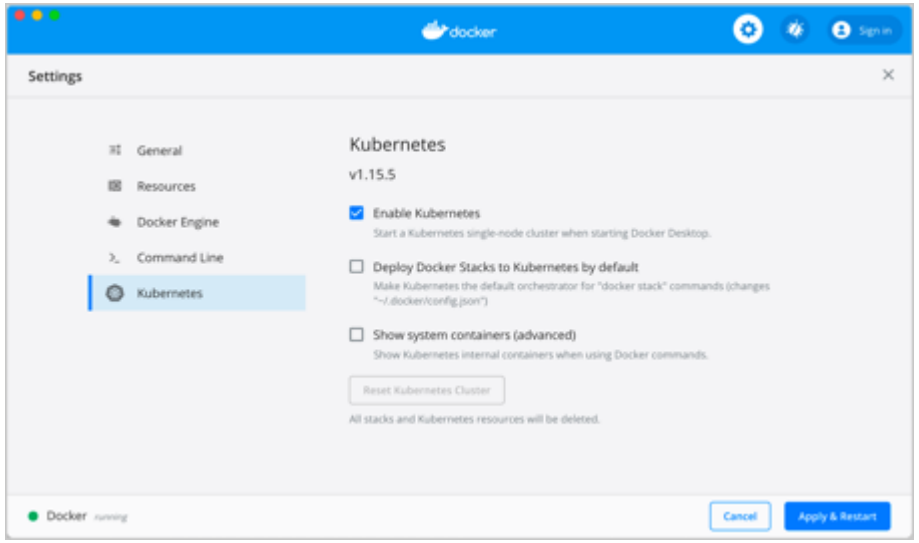


Kubernetes Practice

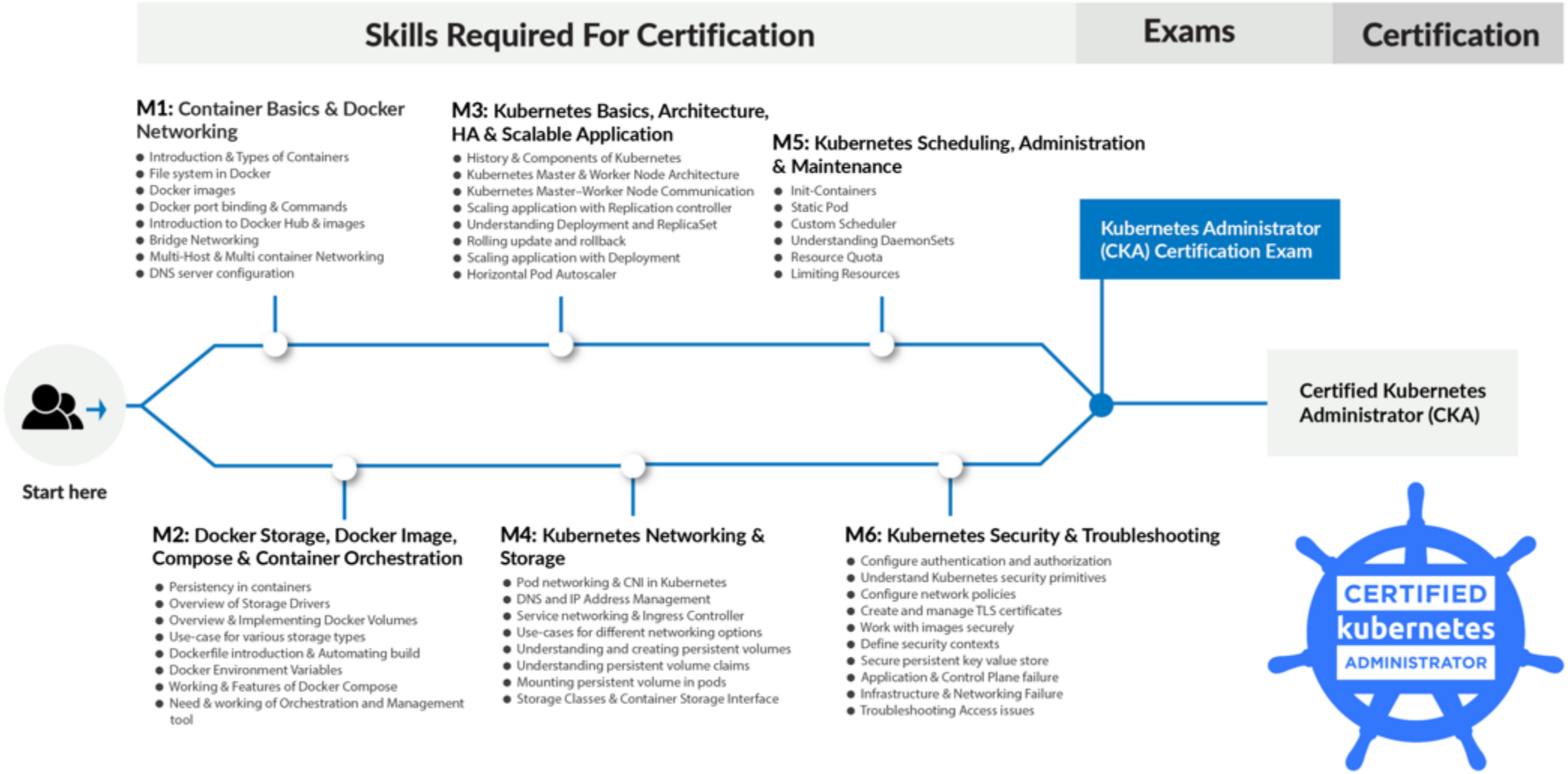
- Minikube



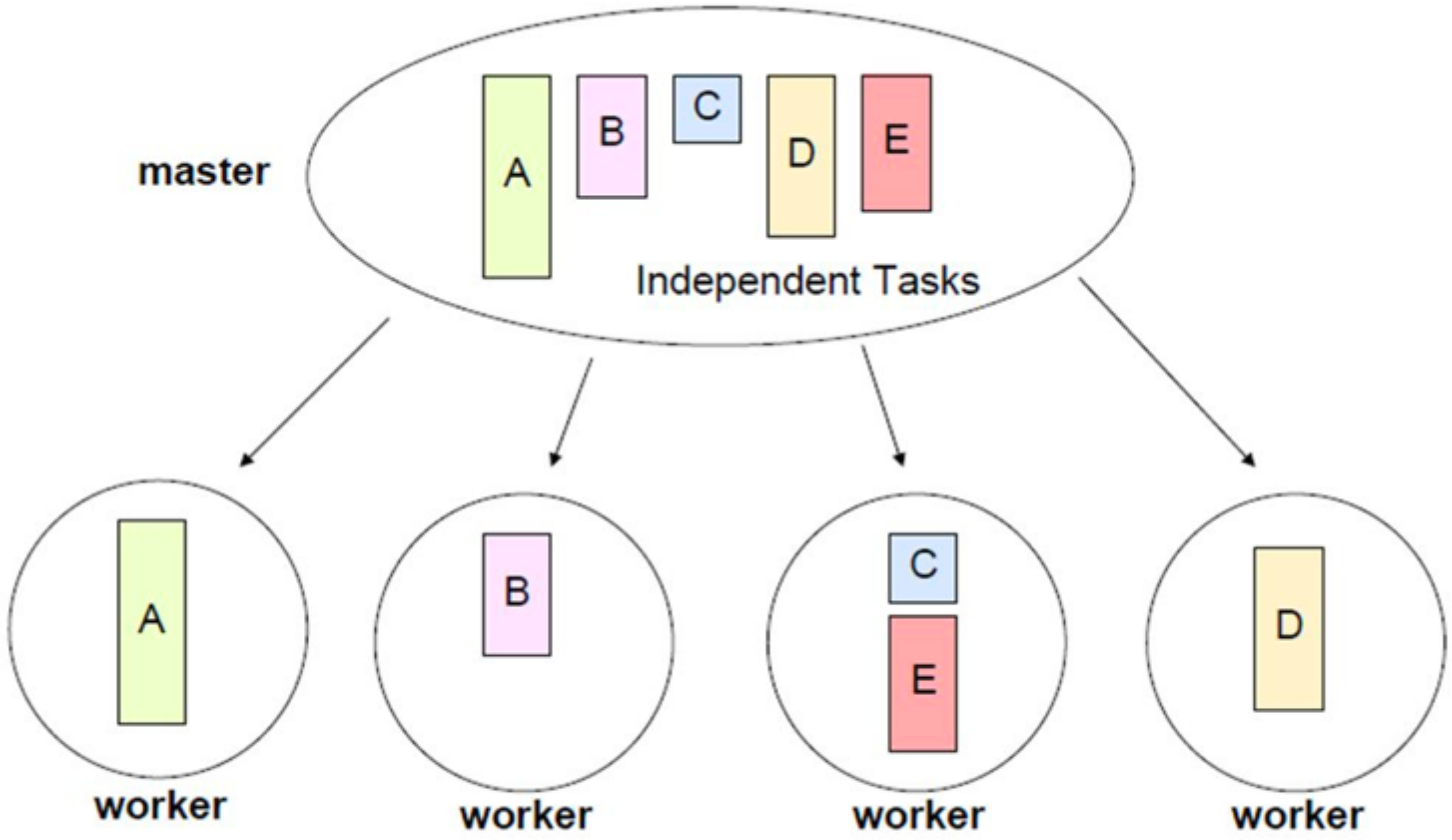
- Katacoda
- Docker desktop
- Cloud
- …



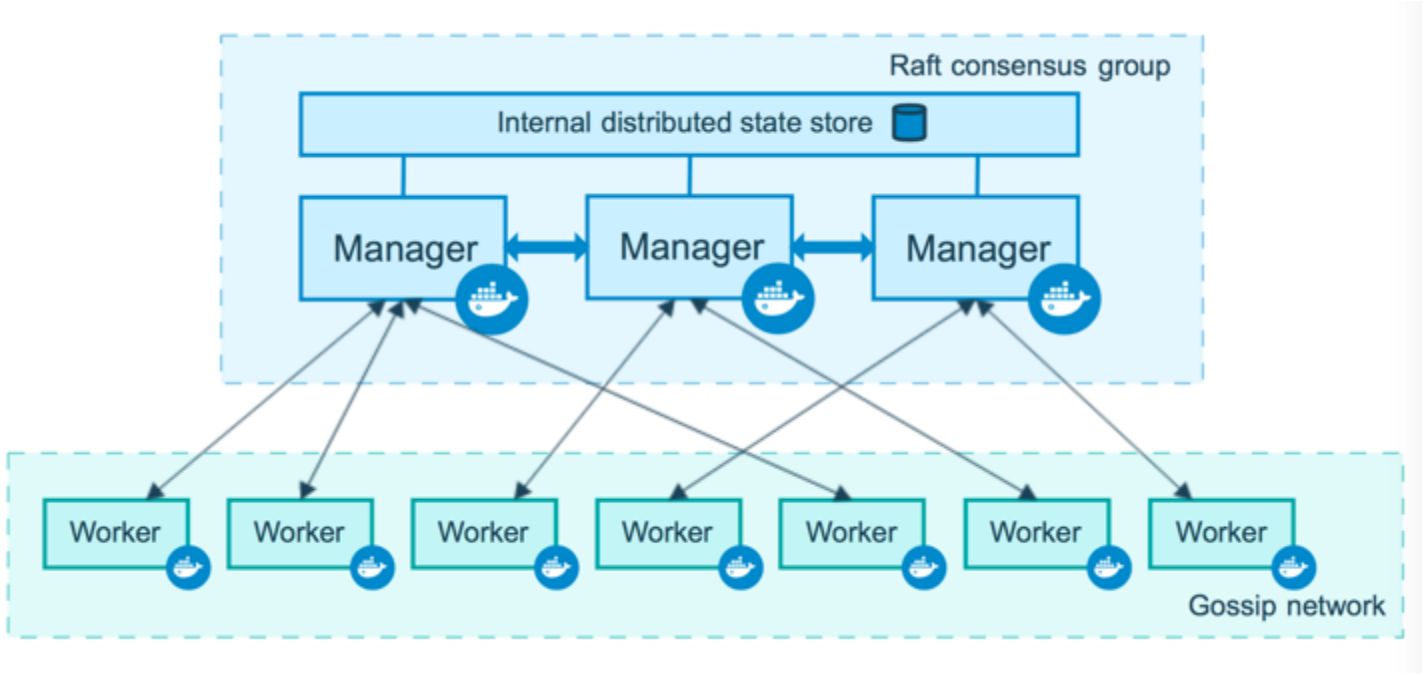
CKA (Certified Kubernetes Administrator)

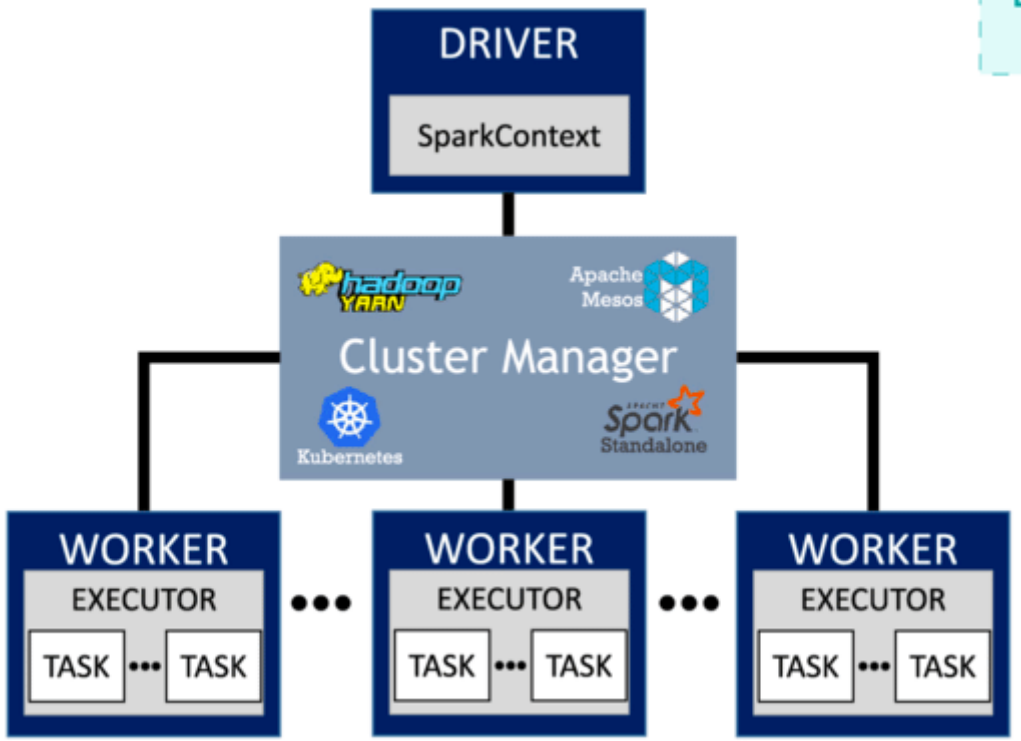


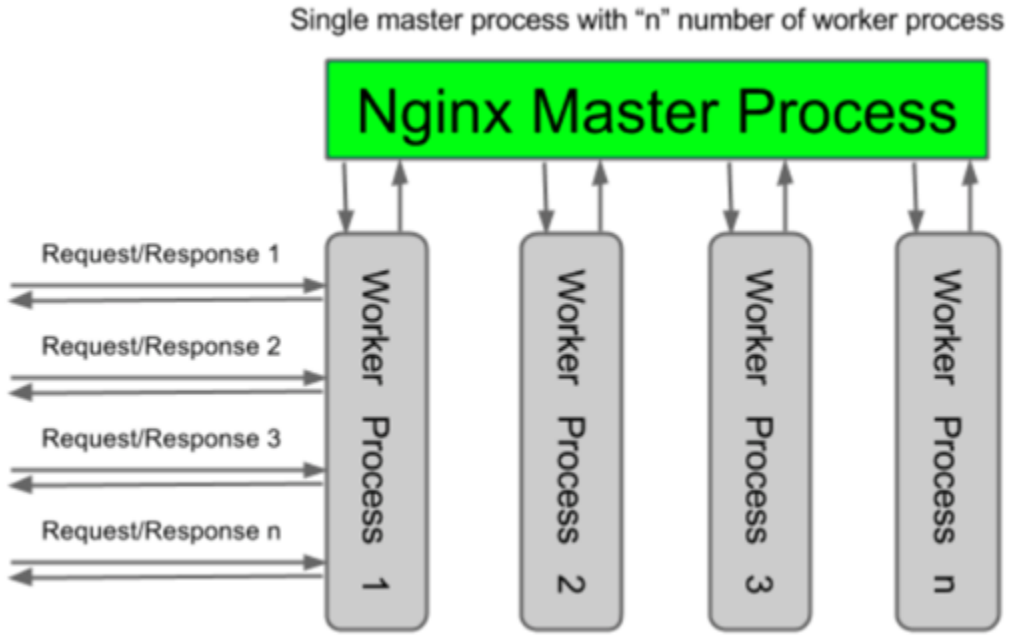
Master-Worker Pattern



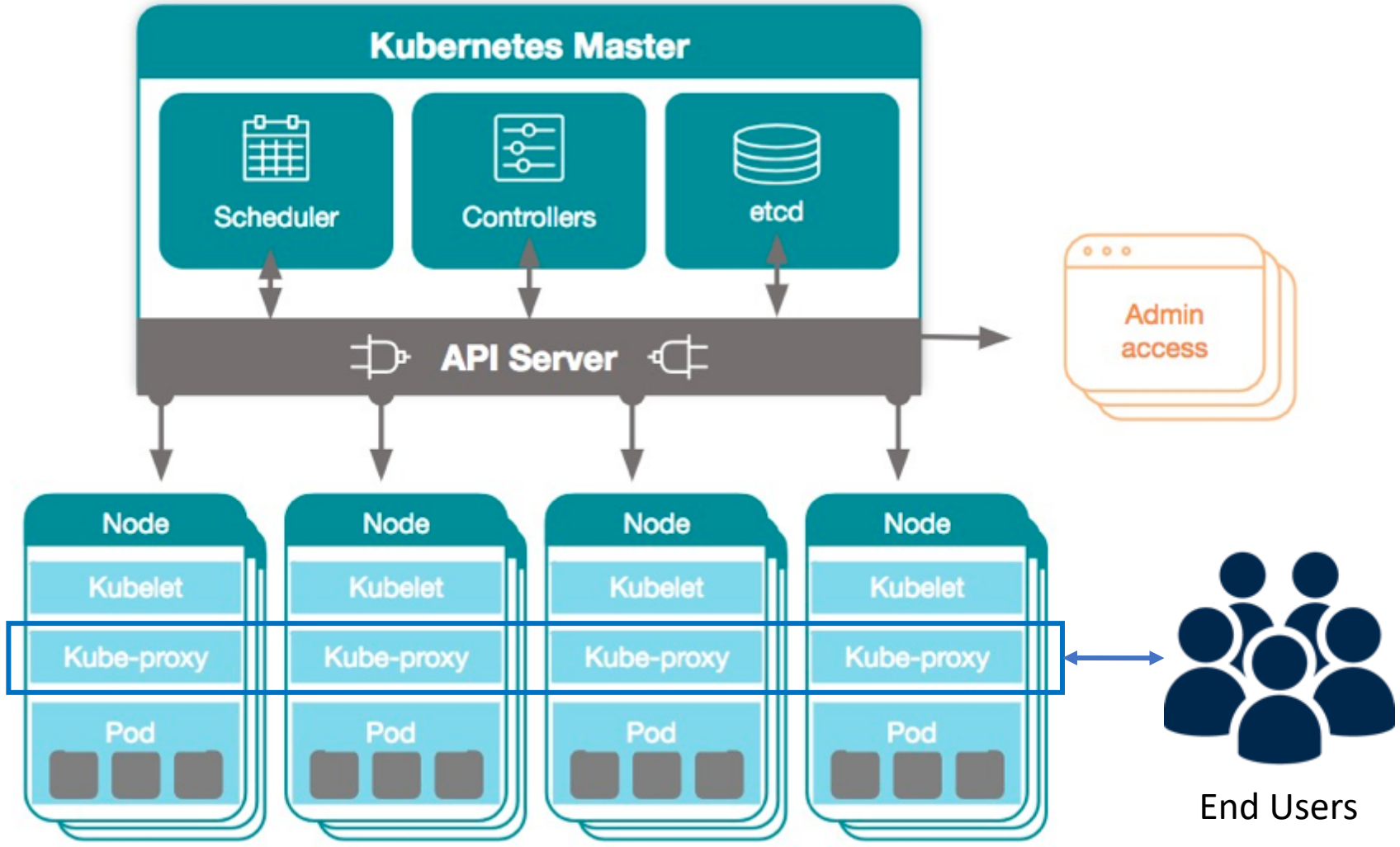
Examples


Raft consensus group
Internal distributed state store
Manager
Manager
Manager
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Gossip network

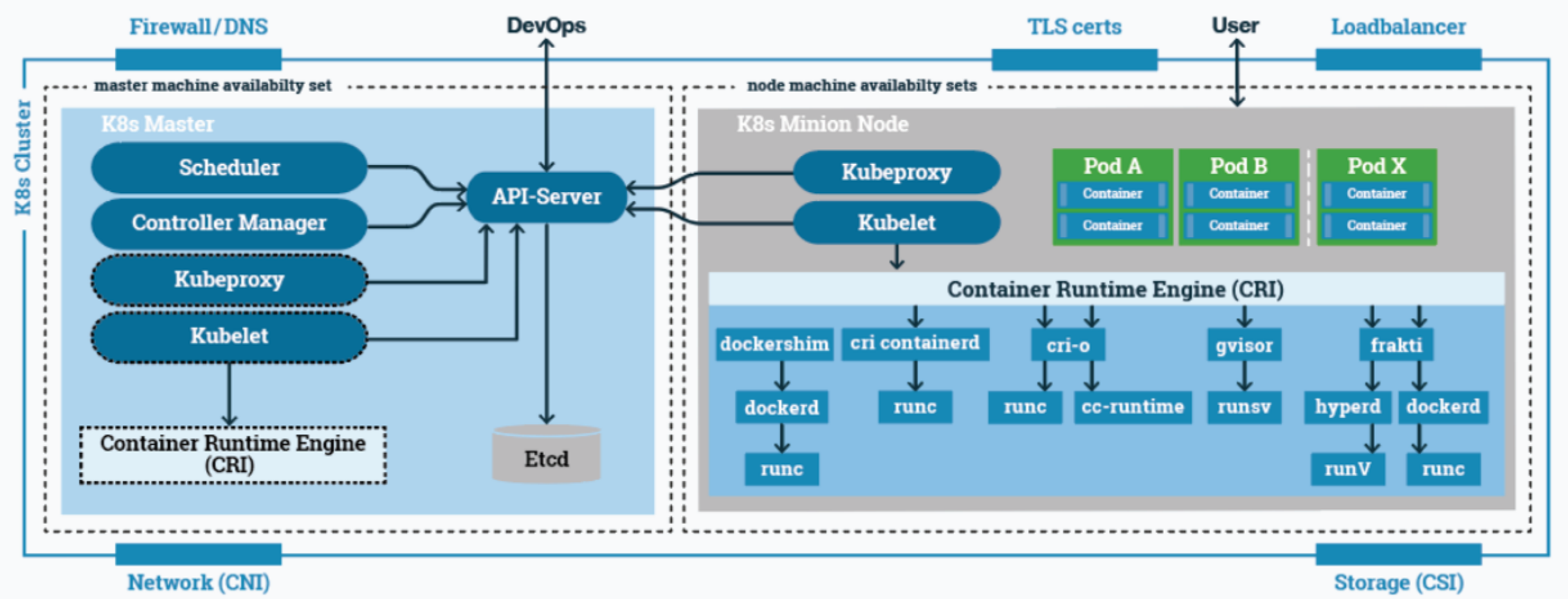

DRIVER
SparkContext
hadoop YARN
Apache Mesos
Cluster Manager
Kubernetes
Spark Standalone
WORKER
EXECUTOR
TASK
TASK
WORKER
EXECUTOR
TASK
TASK
WORKER
EXECUTOR
TASK
TASK


Single master process with "n" number of worker process
Nginx Master Process
Request/Response 1
Request/Response 2
Request/Response 3
Request/Response n
Worker Process 1
Worker Process 2
Worker Process 3
Worker Process n

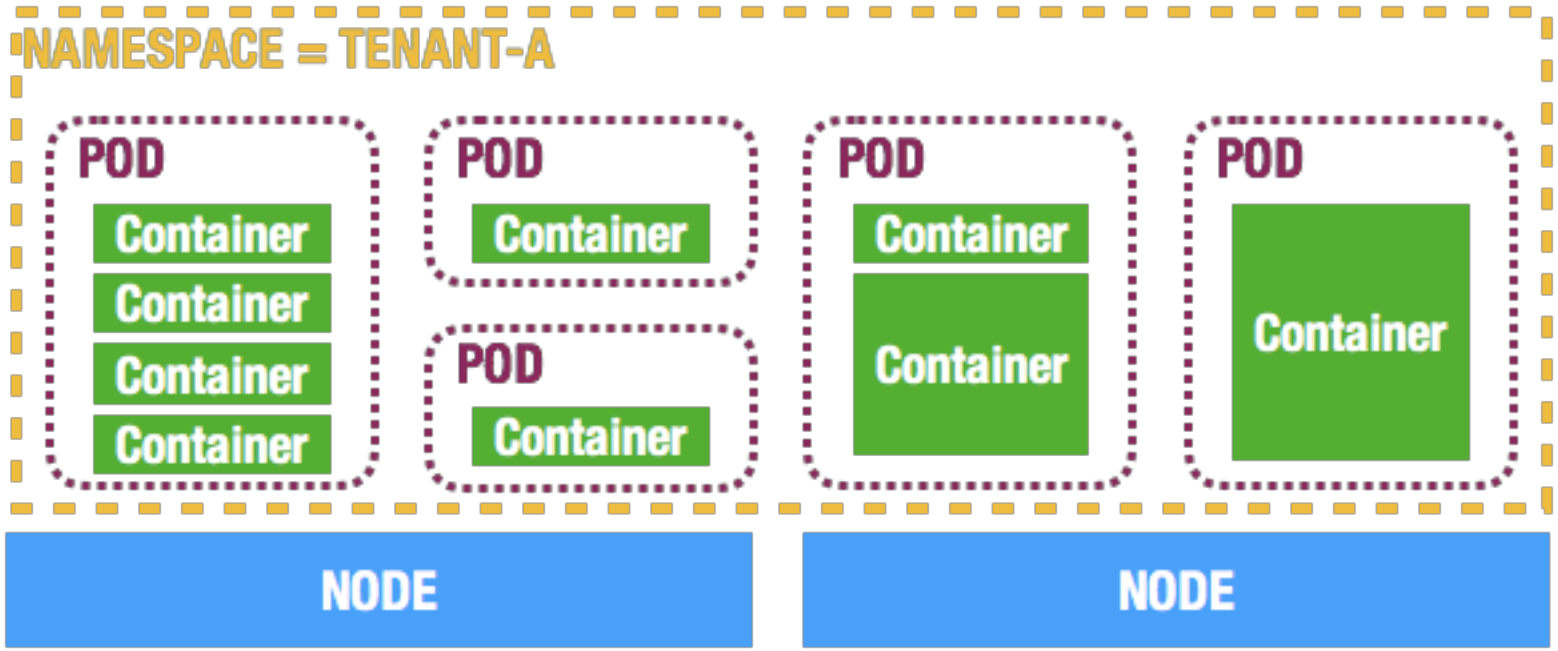
Kubernetes Architecture



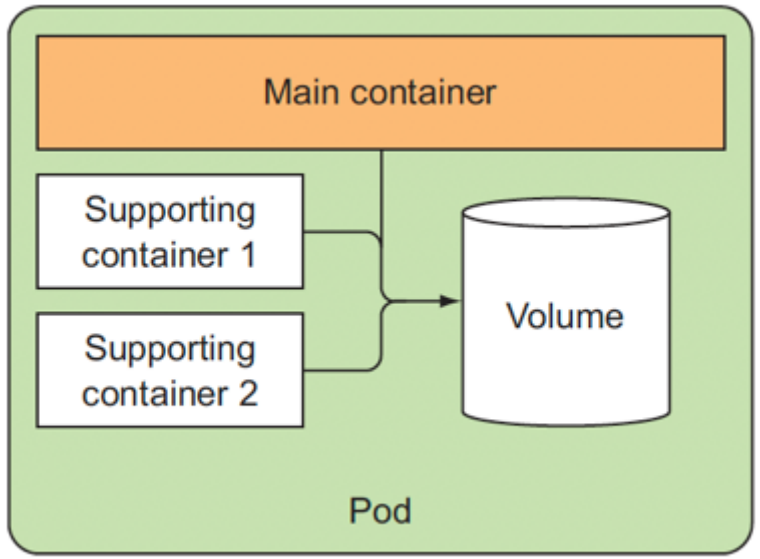
In-Depth Architecture

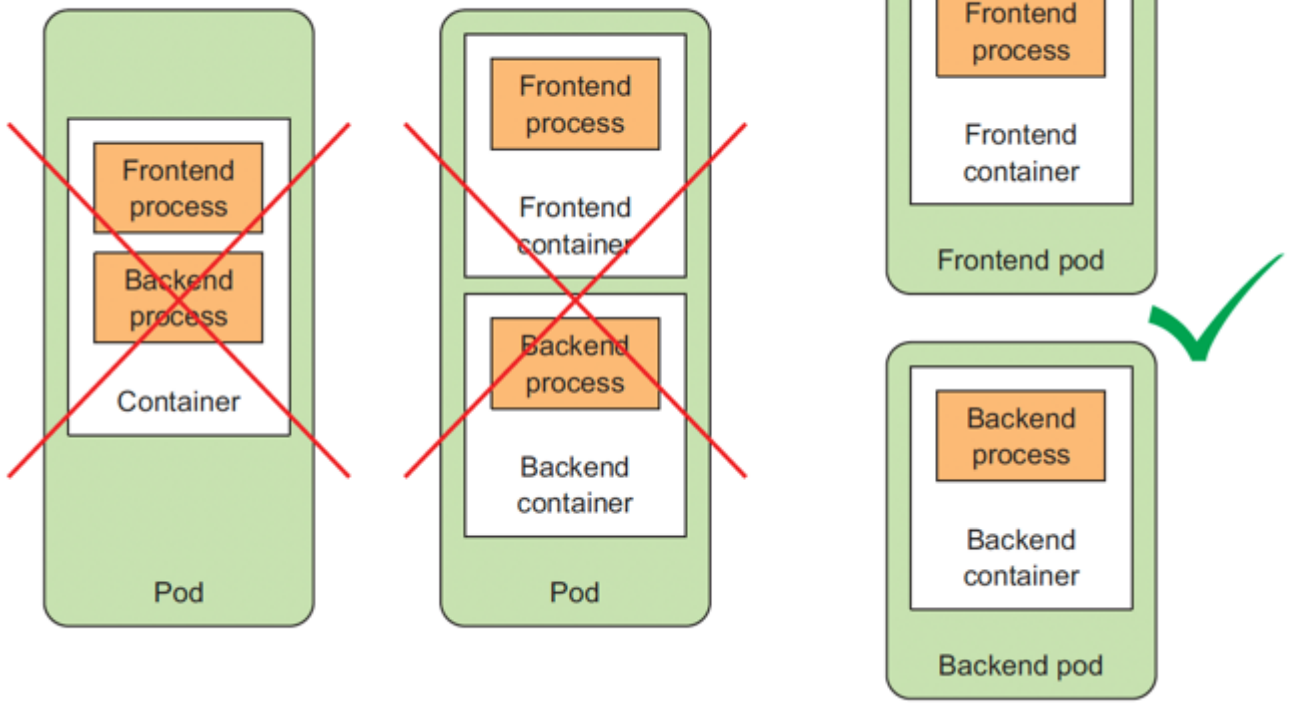


Pod and Container

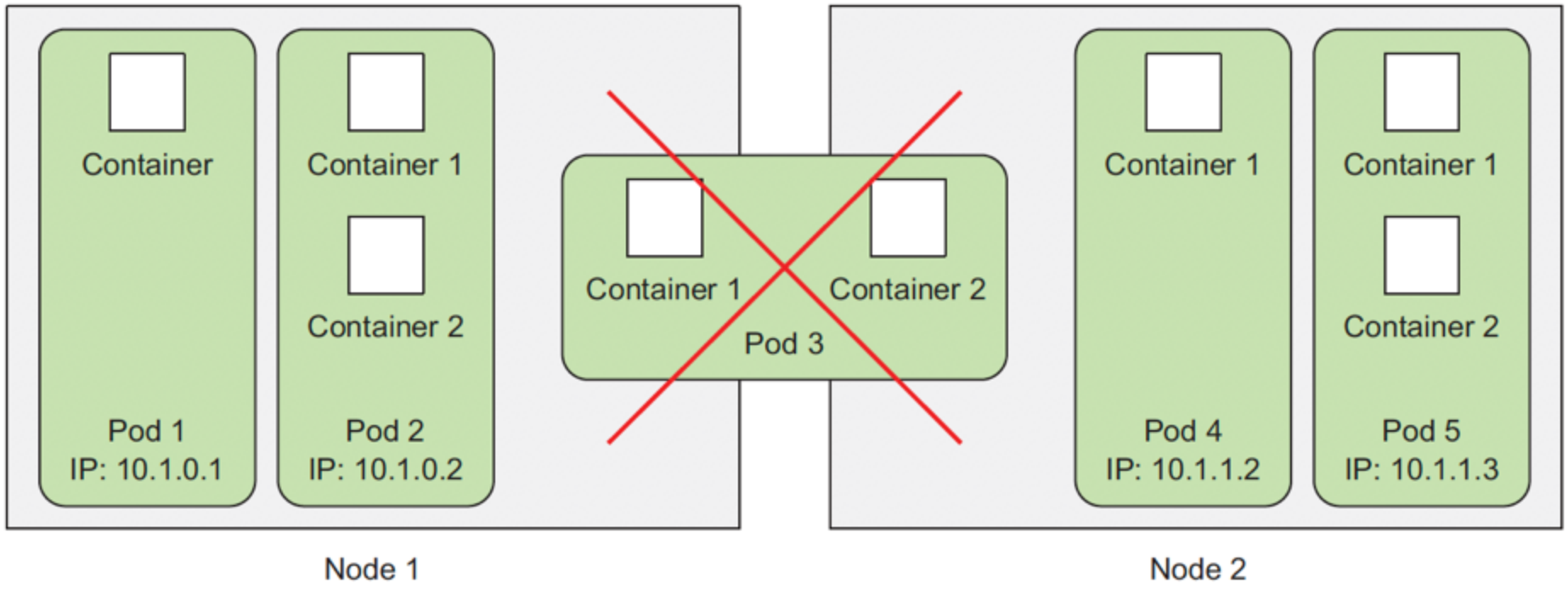


Pod Design

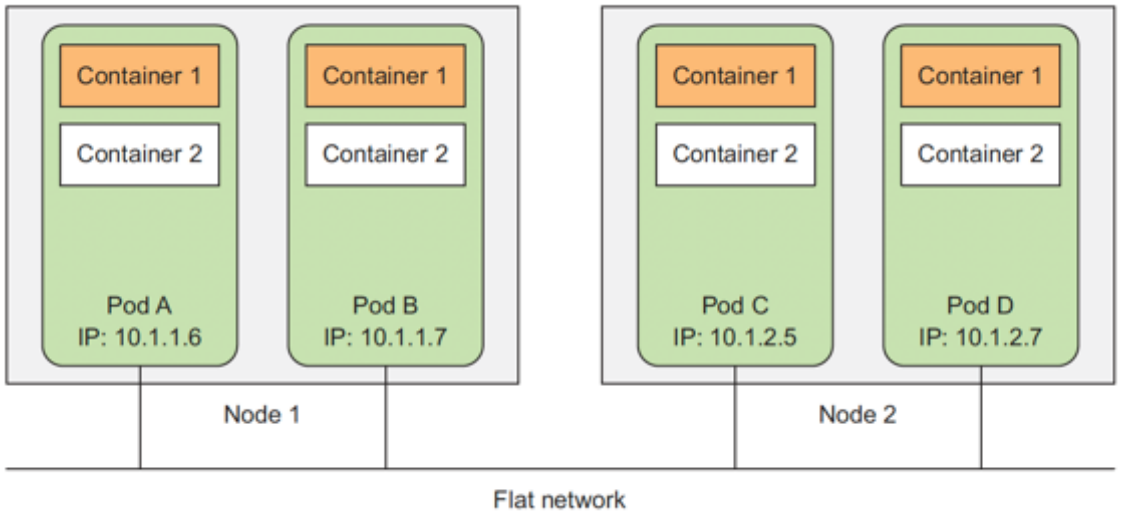

Main container
Supporting container 1
Supporting container 2
Volume
Pod

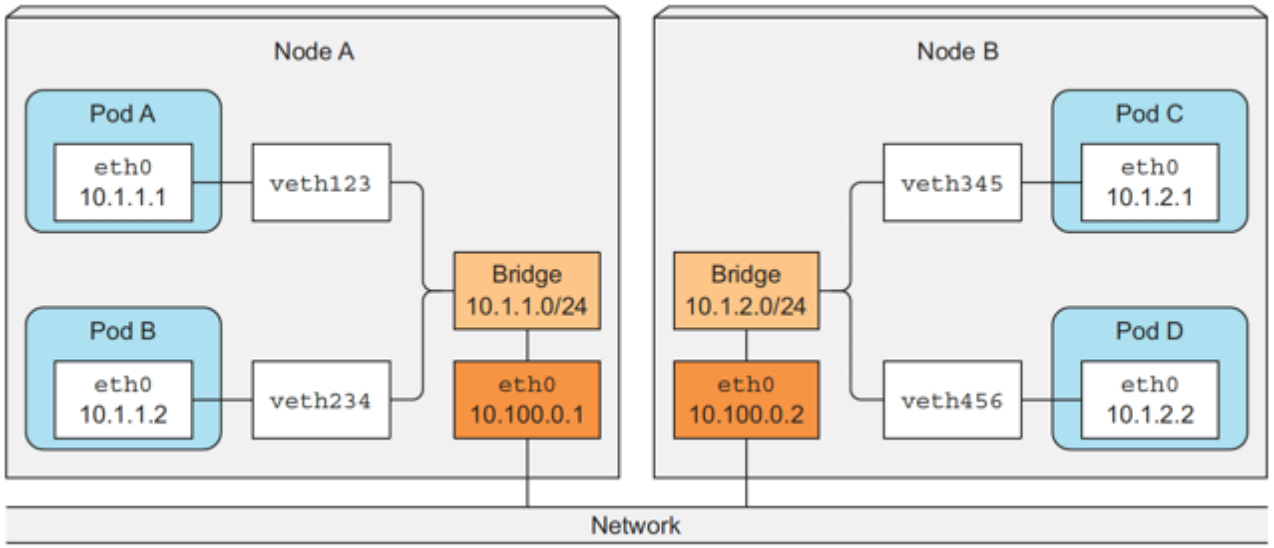

Frontend process
Backend process
Container
Pod
Frontend process
Frontend container
Backend process
Backend container
Pod
Frontend process
Frontend container
Frontend pod
Backend process
Backend container
Backend pod

Pod in a Node

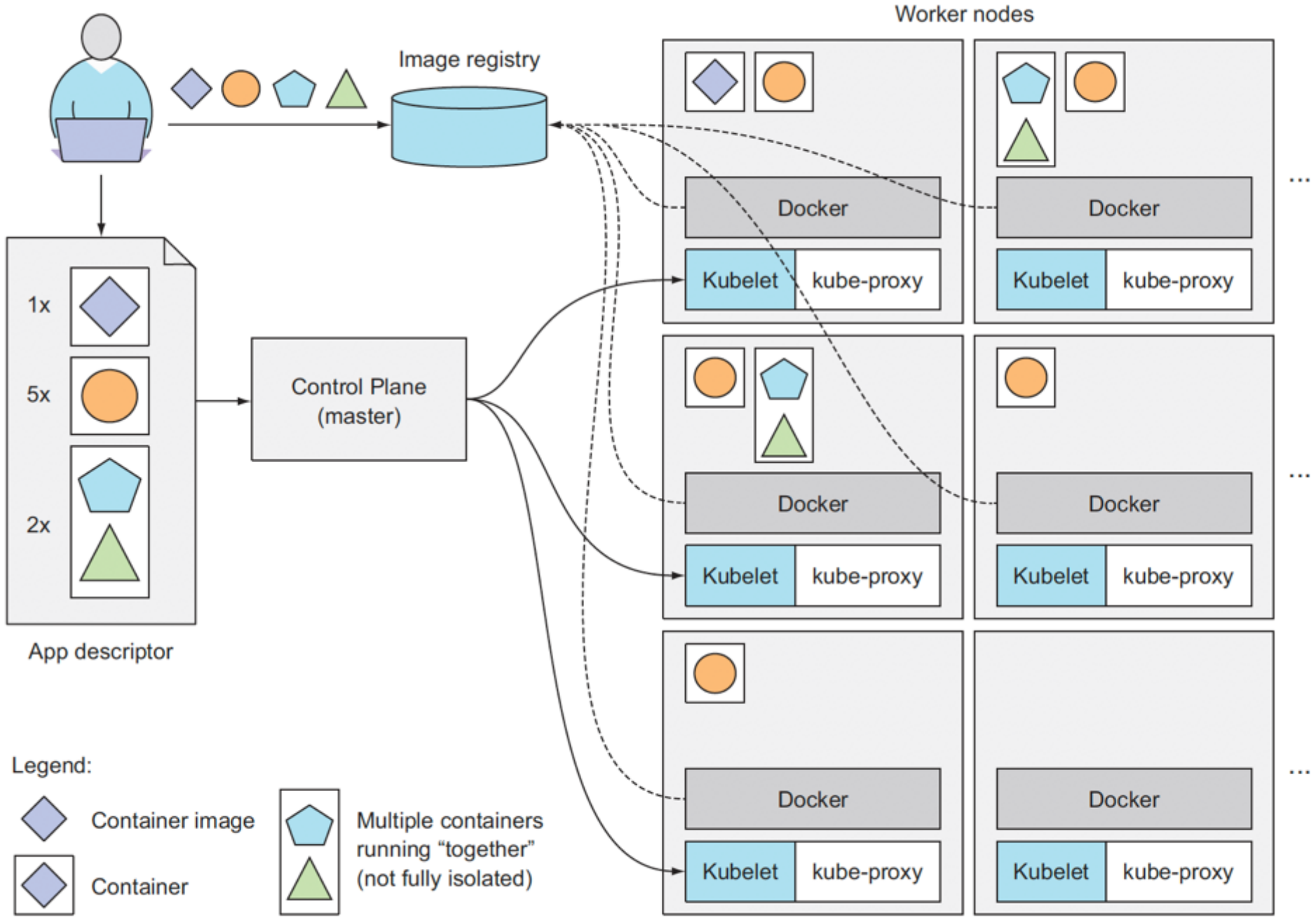

Container
Pod 1
IP: 10.1.0.1
Container 1
Container 2
Pod 2
IP: 10.1.0.2
Container 1
Container 2
Pod 3
Container 1
Container 2
Pod 4
IP: 10.1.1.2
Container 1
Container 2
Pod 5
IP: 10.1.1.3
Node 1
Node 2

Pod IP Address



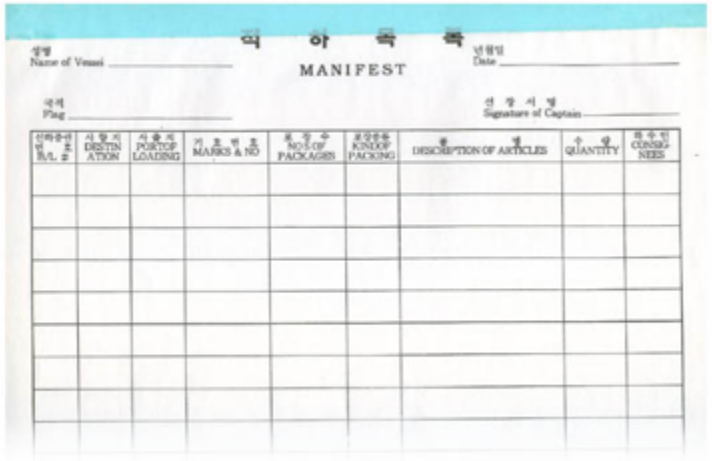


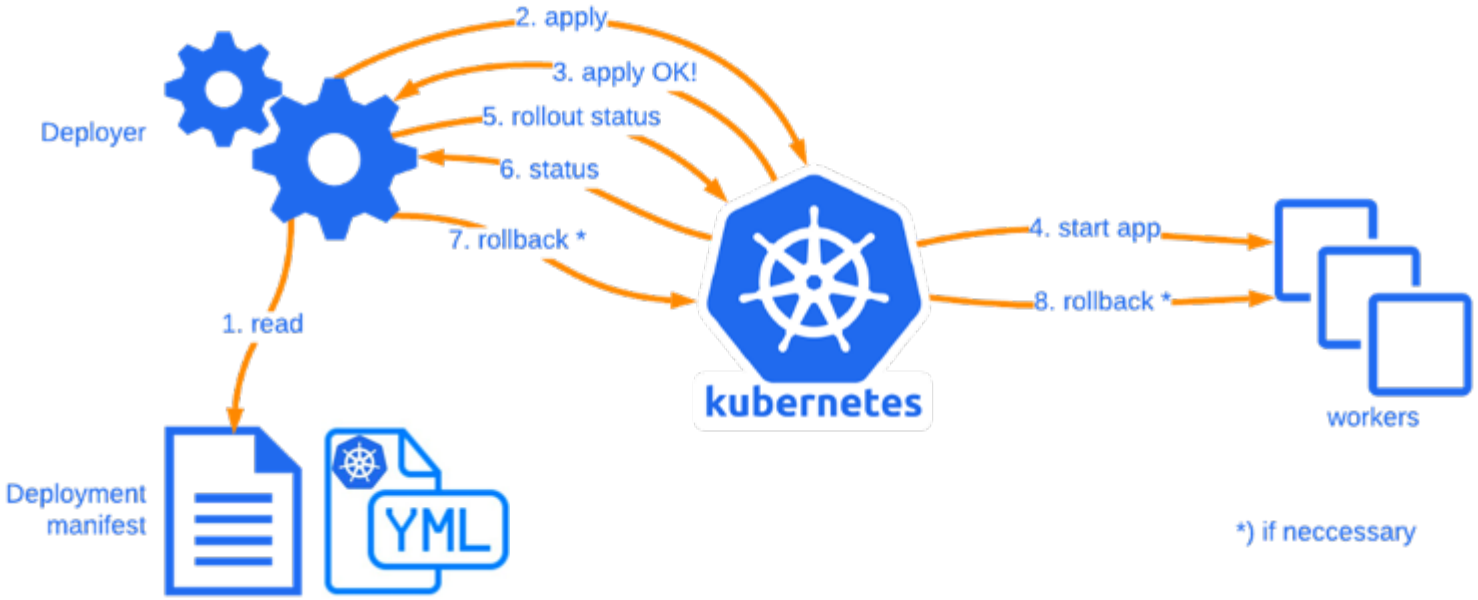
Overview


Worker nodes
Image registry
Docker
Kubelet
kube-proxy
1x
5x
2x
App descriptor
Control Plane
(master)
Legend:
Container image
Container
Multiple containers
running "together"
(not fully isolated)

Manifest

- App descriptor = Manifest



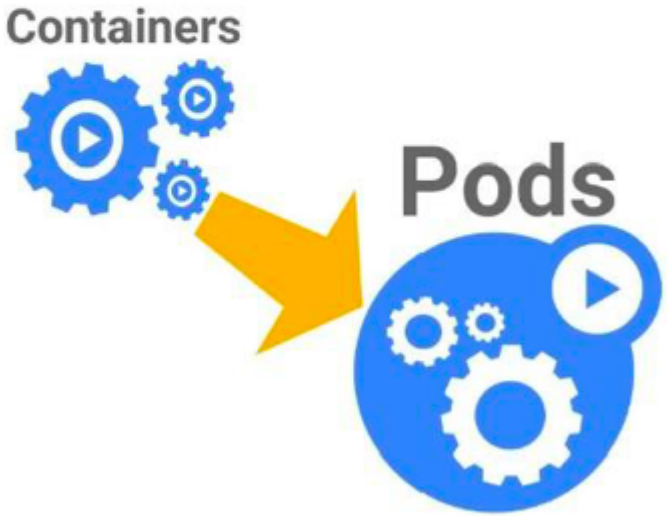


Manifest: YAML File

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    phase: prod
    role: frontend
    name: myfirstpod
  name: myfirstpod
        spec:
          containers:
          - name: filepuller
            image: sliranc/filepuller:latest
            volumeMounts:
            - mountPath: /usr/share/nginx/html
              name: static-vol
          - name: webserver
            image: nginx:latest
            ports:
            - containerPort: 80
            volumeMounts:
              - mountPath: /usr/share/nginx/html
                name: static-vol
          volumes:
          - name: static-vol
            emptyDir: {}
```

kind: System object \ resource
Examples: Pod, RC, Service etc...

Spec: is the specification of the desired state of an **object**.



8 Ways to Create a Pod

Resource	Result	RBAC resource
Pod	Creates Pod	Pods
ReplicationController	Creates Pod with replicas and prevent it from crashing	replicationcontrollers
ReplicaSet		replicasets
Deployment	Creates Pod and replicaset	deployments
DaemonSet	Creates Pod in each node in the cluster	daemonsets
Job	Creates Pod to completion	jobs
CronJob	Creates Pod to completion periodically	cronjobs
StatefulSet	Creates Pod with order	statefulsets

Label

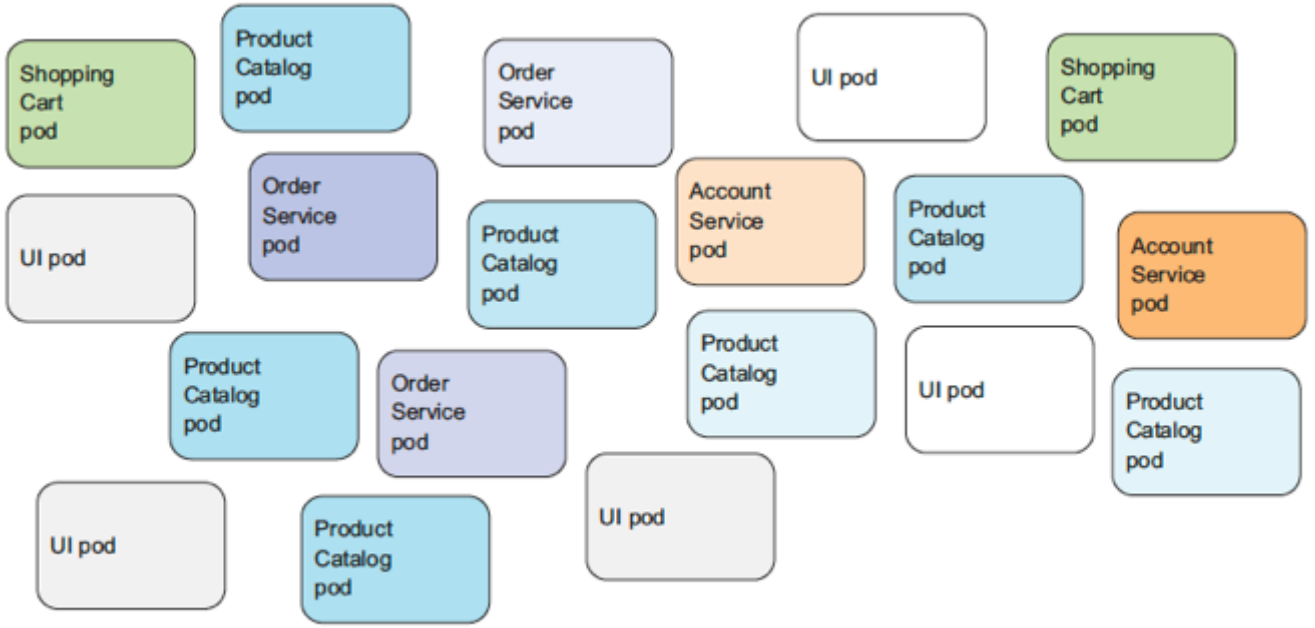
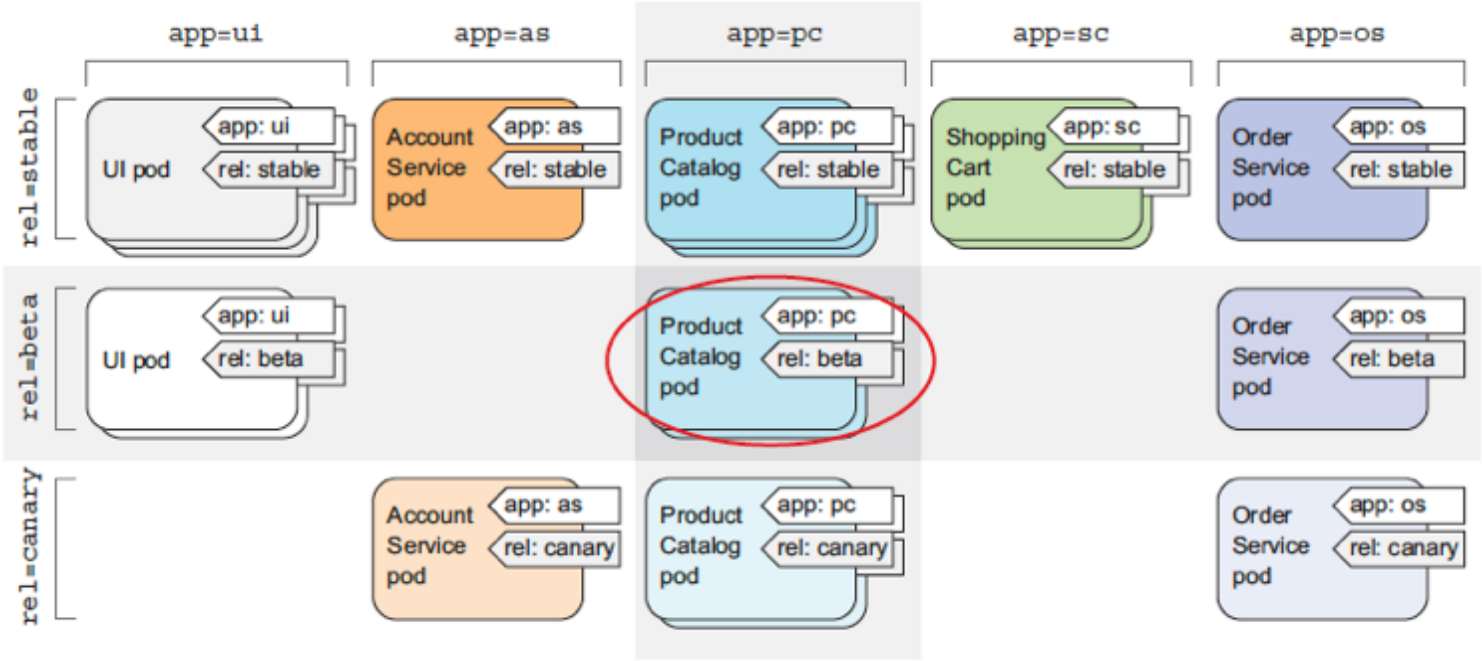


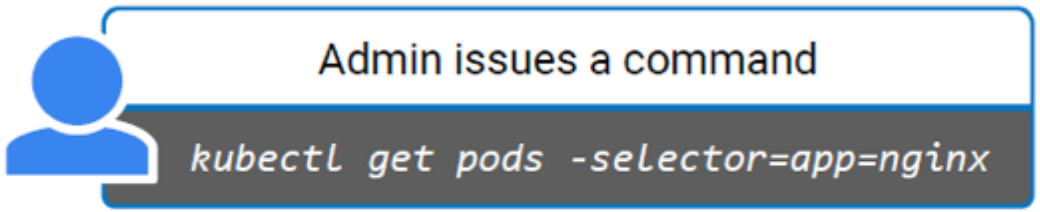
Figure 3.6 Uncategorized pods in a microservices architecture



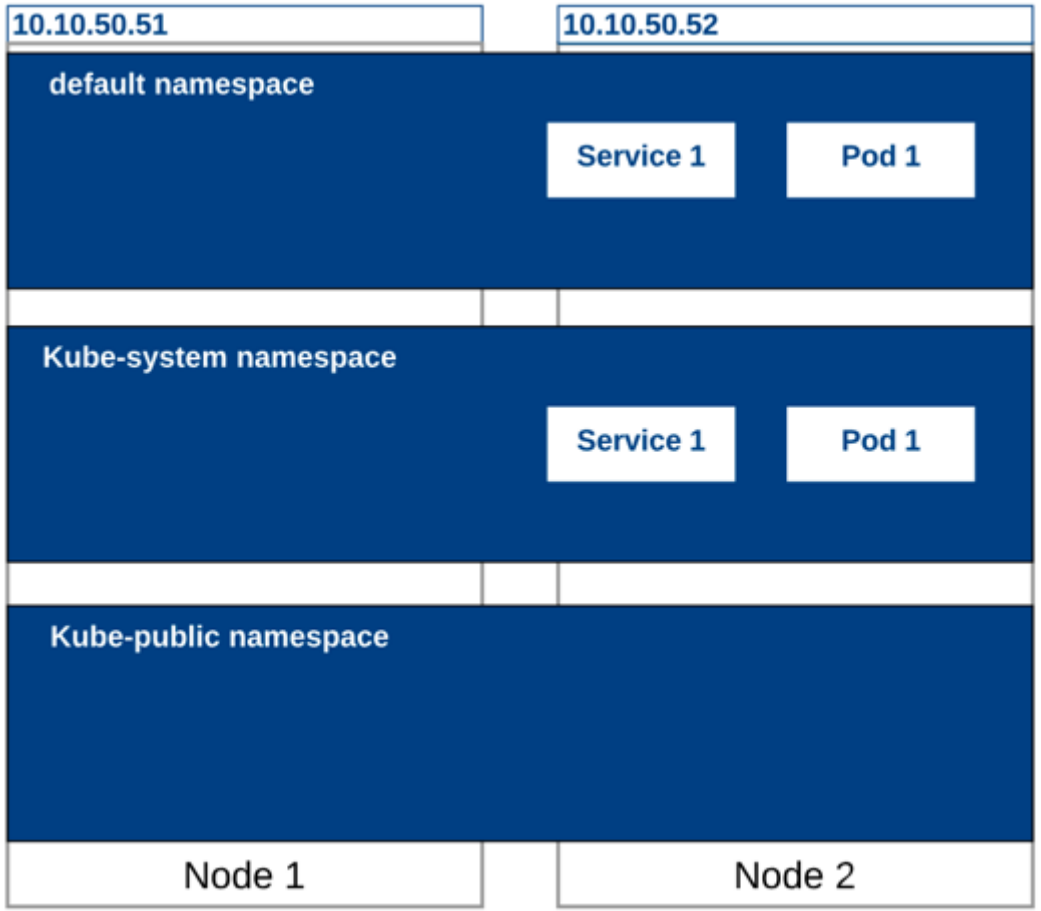
Label-Selector

Labels can be matched by label selectors

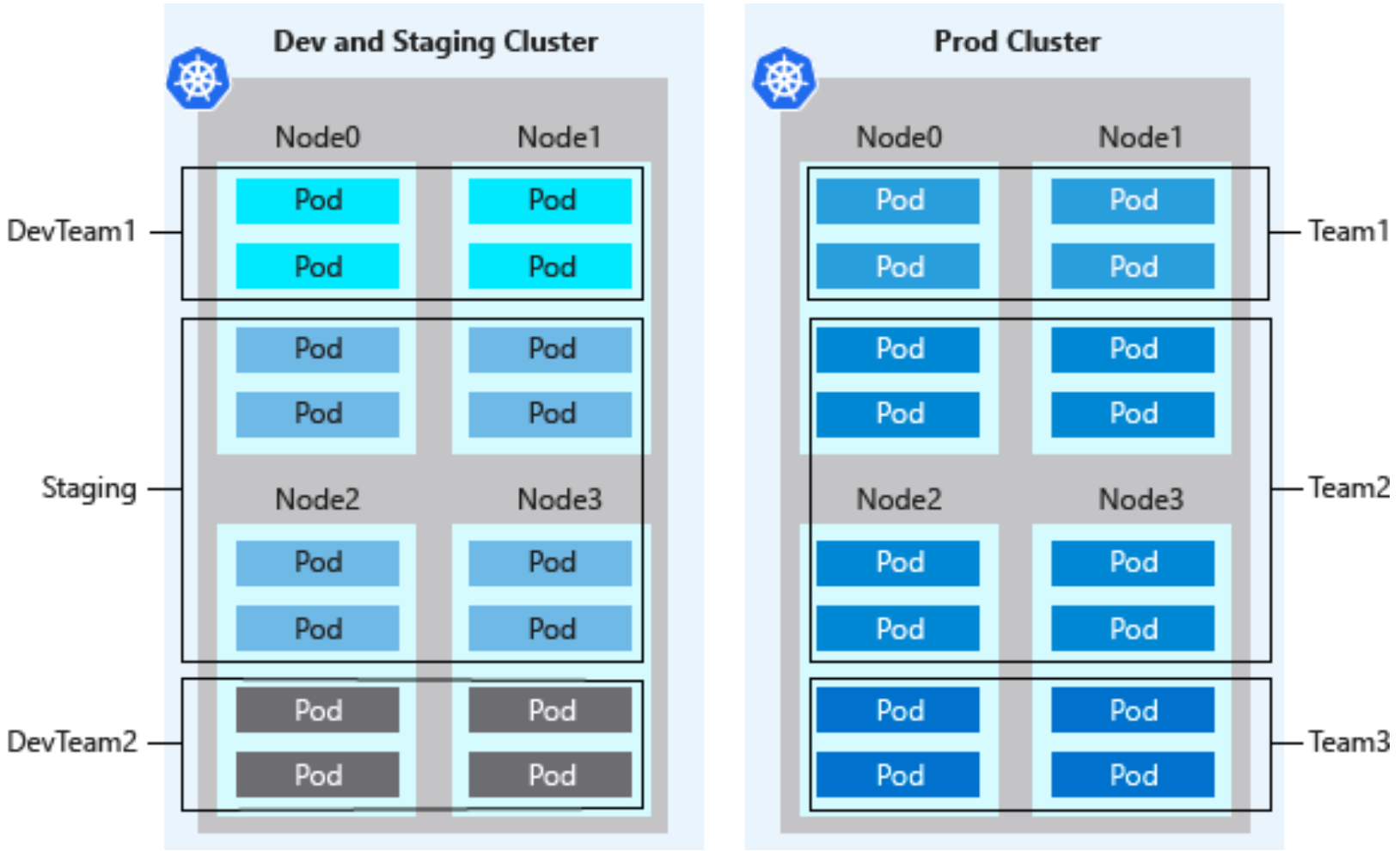
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
    env: dev
    stack: frontend
spec:
  replicas: 3
  selector:
    matchLabels
    app: nginx
```

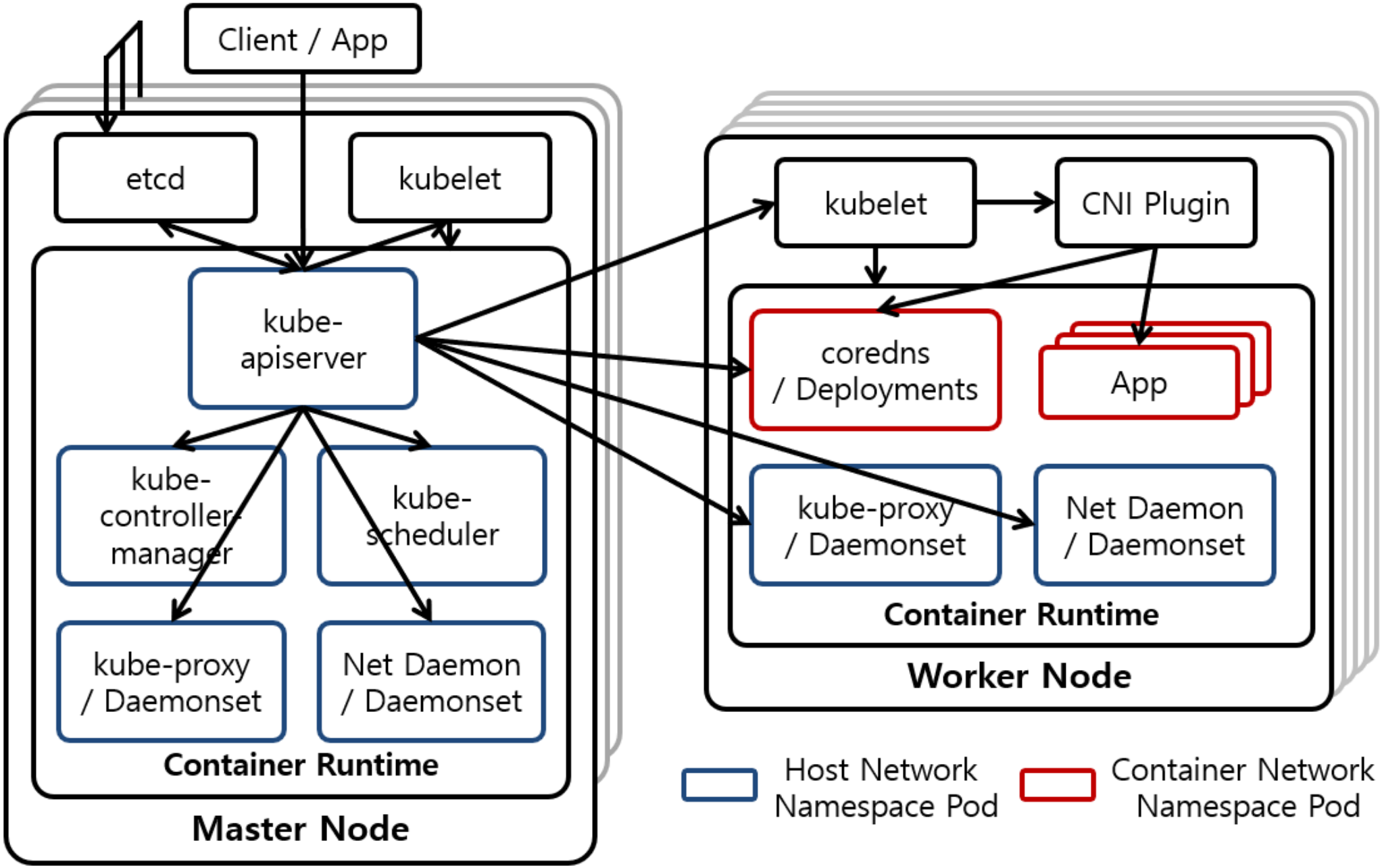


Namespace, Node, Pod



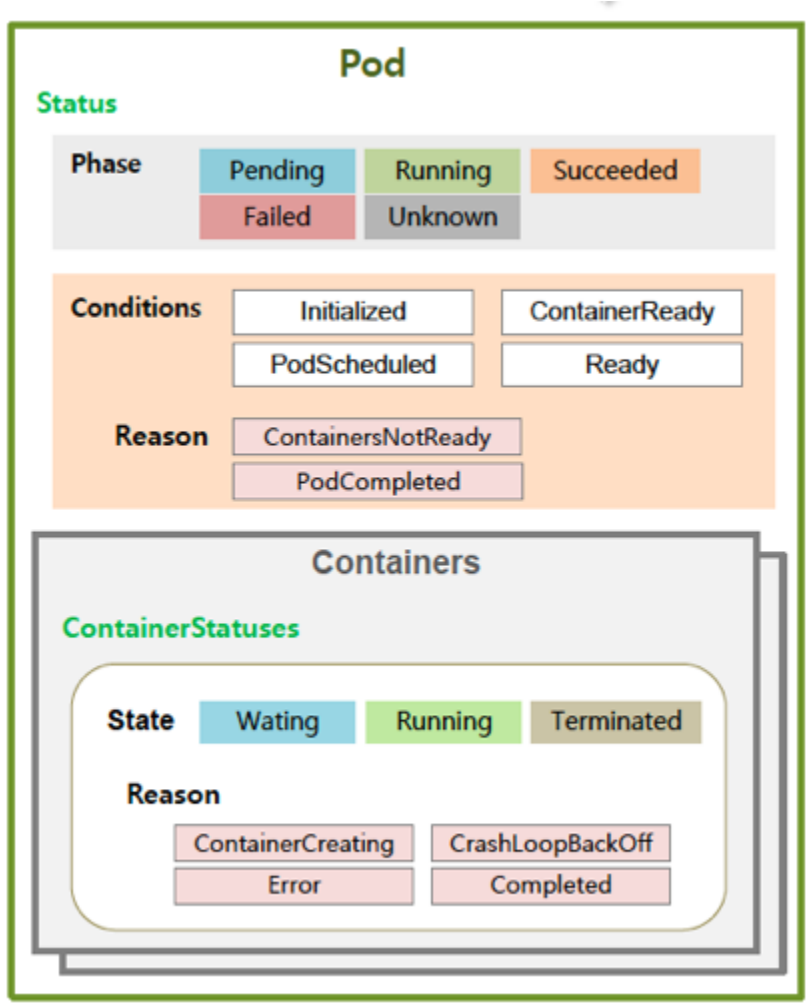
Namespace Best Practice



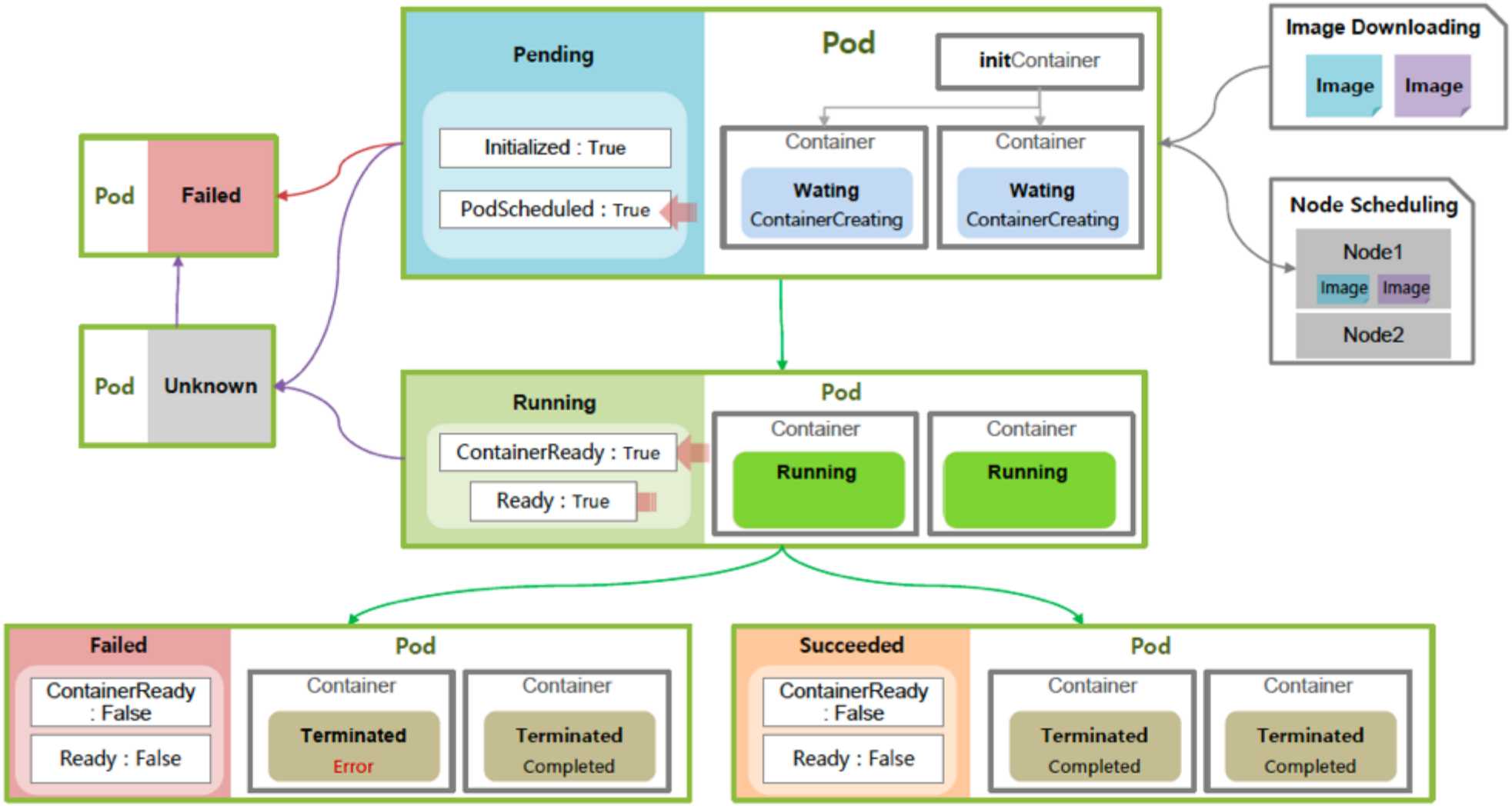

Client / App
etcd
kubelet
kube-
apiserver
kube-
controller-
manager
kube-
scheduler
kube-proxy
/ Daemonset
Net Daemon
/ Daemonset
Container Runtime
Master Node
kubelet
CNI Plugin
coredns
/ Deployments
App
kube-proxy
/ Daemonset
Net Daemon
/ Daemonset
Container Runtime
Worker Node
Host Network
Namespace Pod
Container Network
Namespace Pod

Pod, Container Status

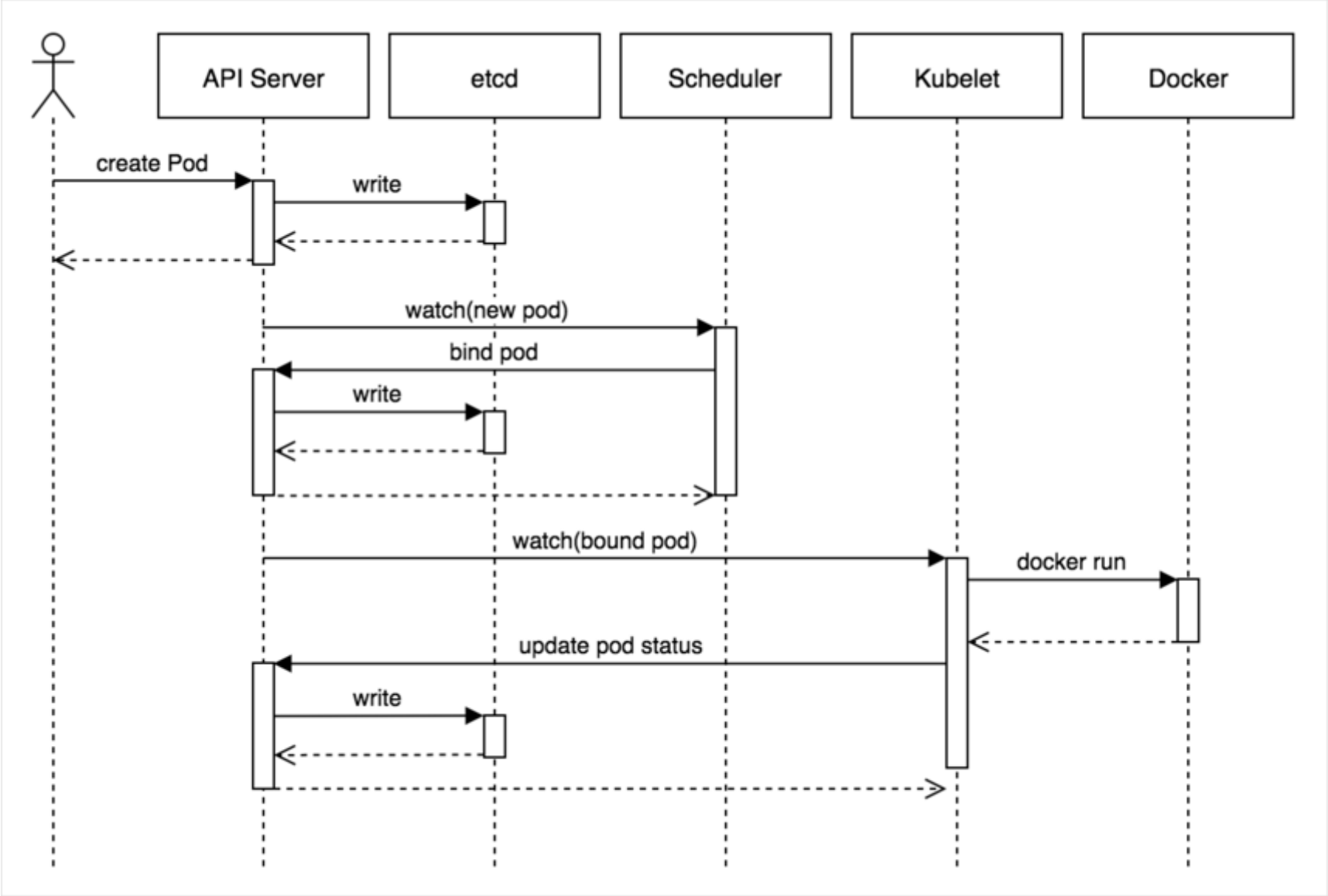
```
status:
  phase: Pending
  conditions:
    - type: Initialized
      status: 'True'
      lastProbeTime: null
      lastTransitionTime: '2019-09-26T22:07:56Z'

    - type: PodScheduled
      status: 'True'
      lastProbeTime: null
      lastTransitionTime: '2019-09-26T22:07:56Z'

    - type: ContainersReady
      status: 'False'
      lastProbeTime: null
      lastTransitionTime: '2019-09-26T22:08:11Z'
      reason: ContainersNotReady
    - type: Ready
      status: 'False'
      lastProbeTime: null
      lastTransitionTime: '2019-09-26T22:08:11Z'
      reason: ContainersNotReady

  containerStatuses:
    - name: container
      state:
        waiting:
          reason: ContainerCreating
      lastState: {}
      ready: false
      restartCount: 0
      image: tmkube/init
      imageID: ''
      started: false
```



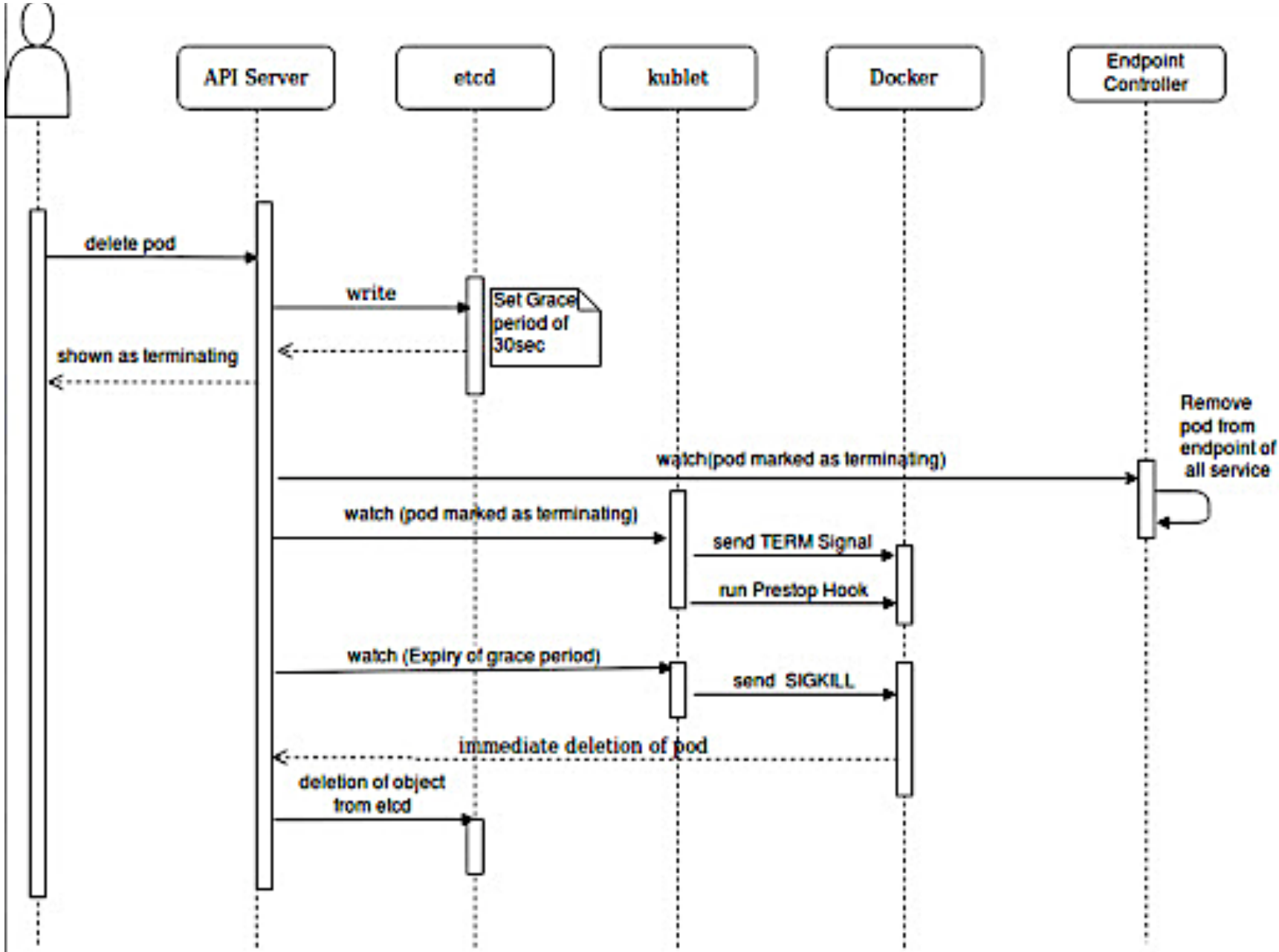
Pod Lifecycle



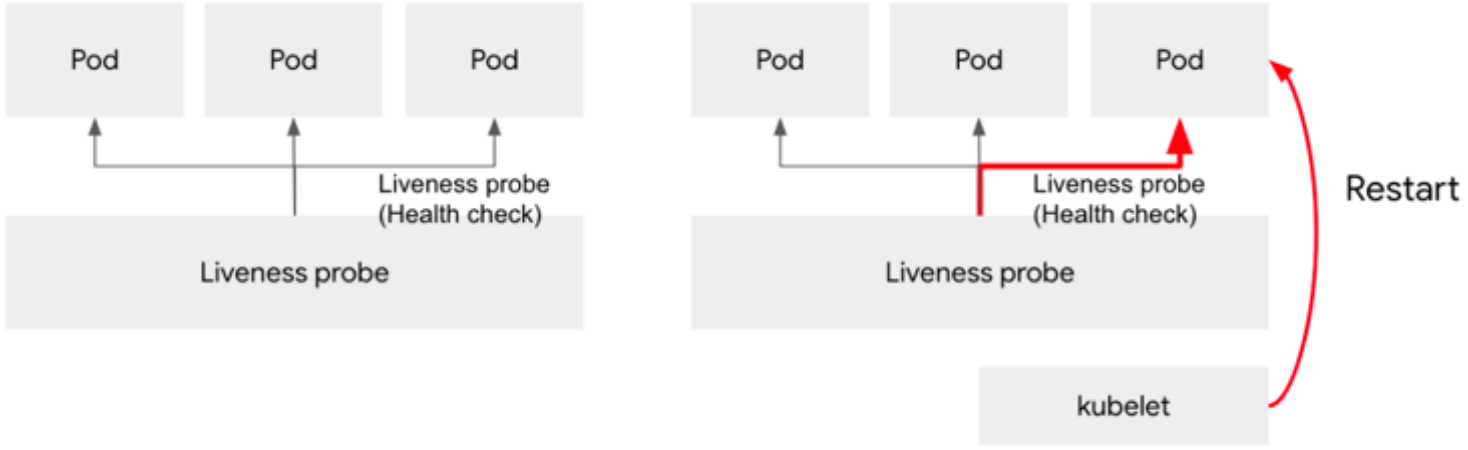
Create Pod

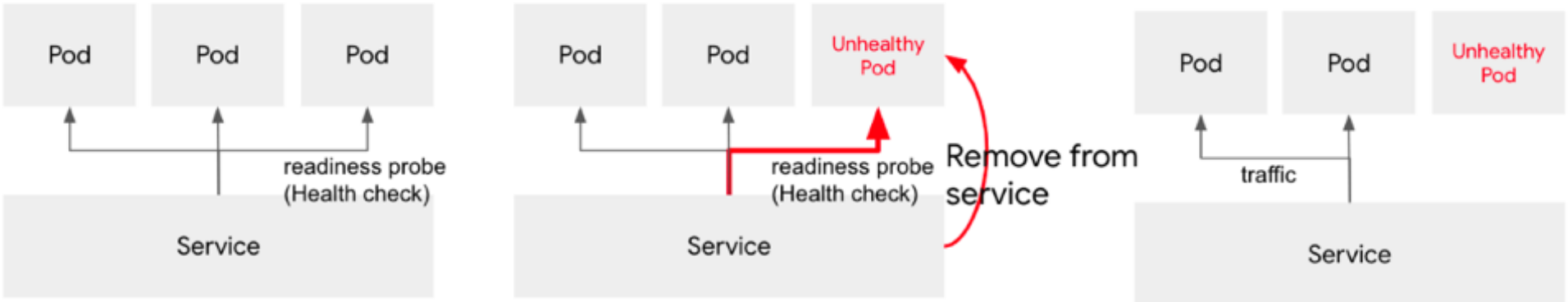

API Server
etcd
Scheduler
Kubelet
Docker
create Pod
write
watch(new pod)
bind pod
write
watch(bound pod)
docker run
update pod status
write

Delete Pod

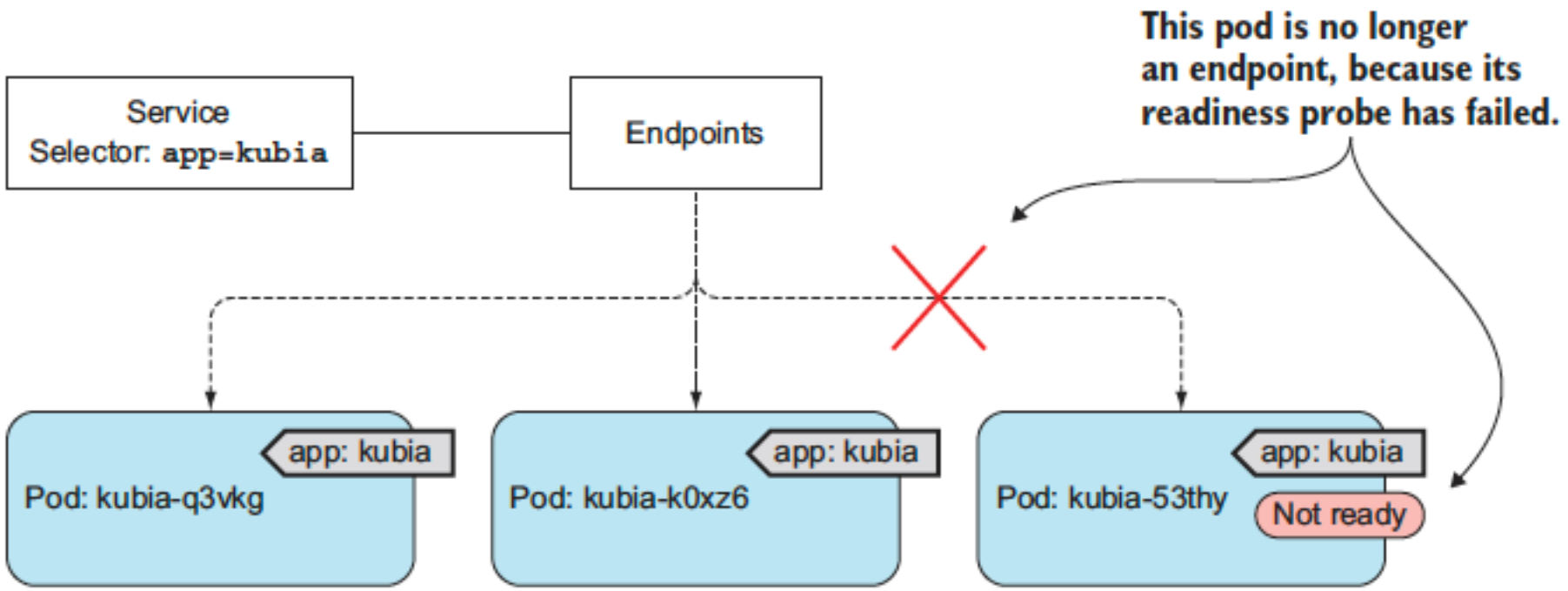

API Server
etcd
kublet
Docker
Endpoint Controller
delete pod
write
Set Grace period of 30sec
shown as terminating
Remove pod from endpoint of all service
watch(pod marked as terminating)
watch (pod marked as terminating)
send TERM Signal
run Prestop Hook
watch (Expiry of grace period)
send SIGKILL
immediate deletion of pod
deletion of object from etcd

Liveness Probe & Readiness Probe

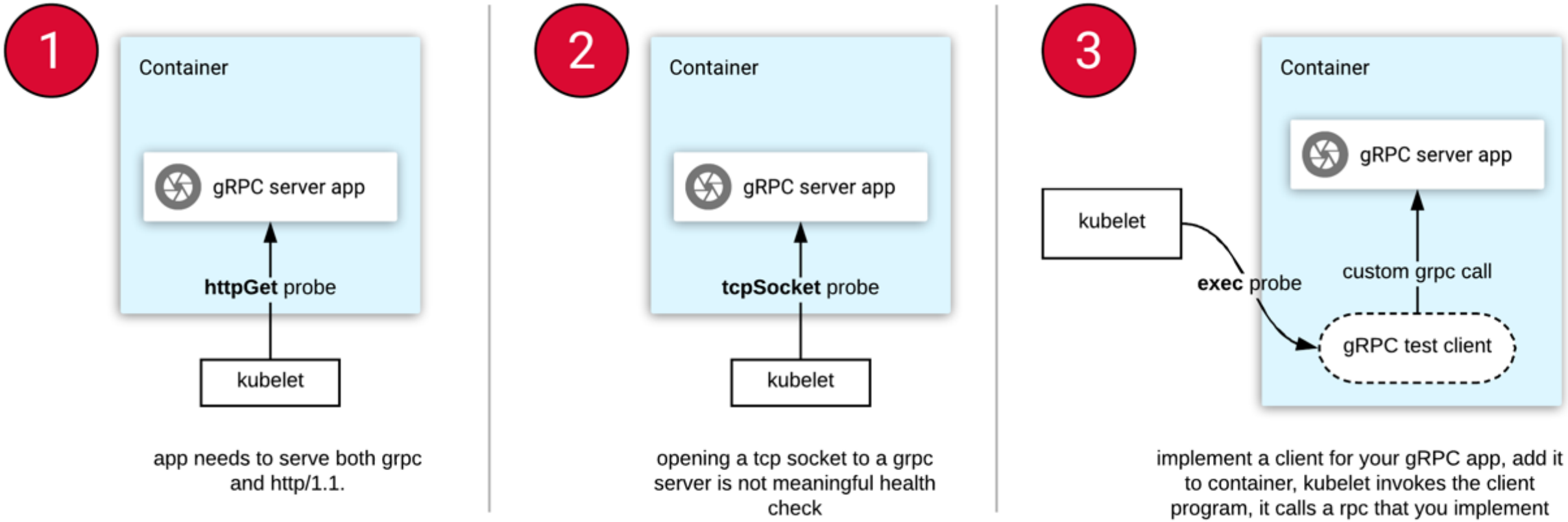




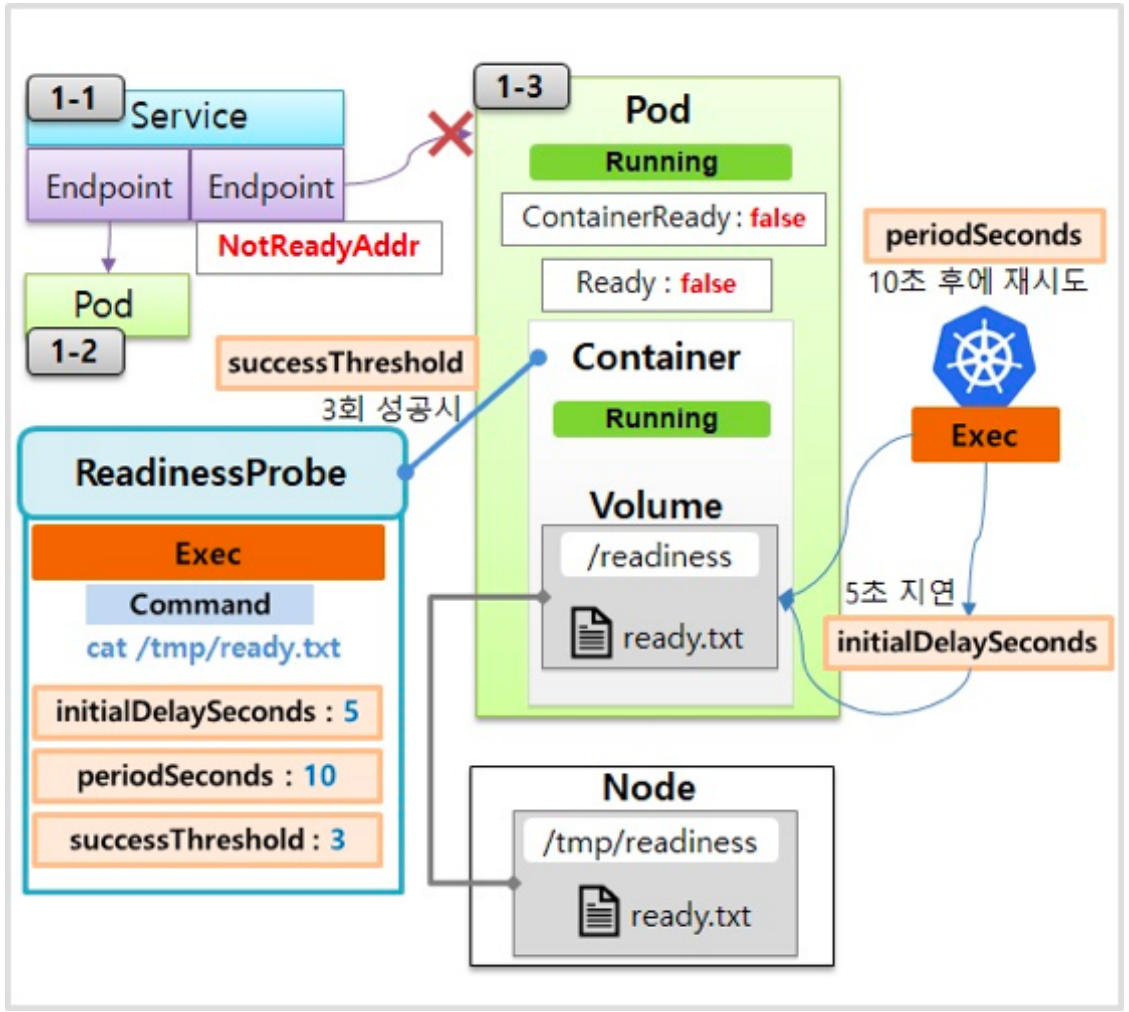
Readiness, Service, Endpoints

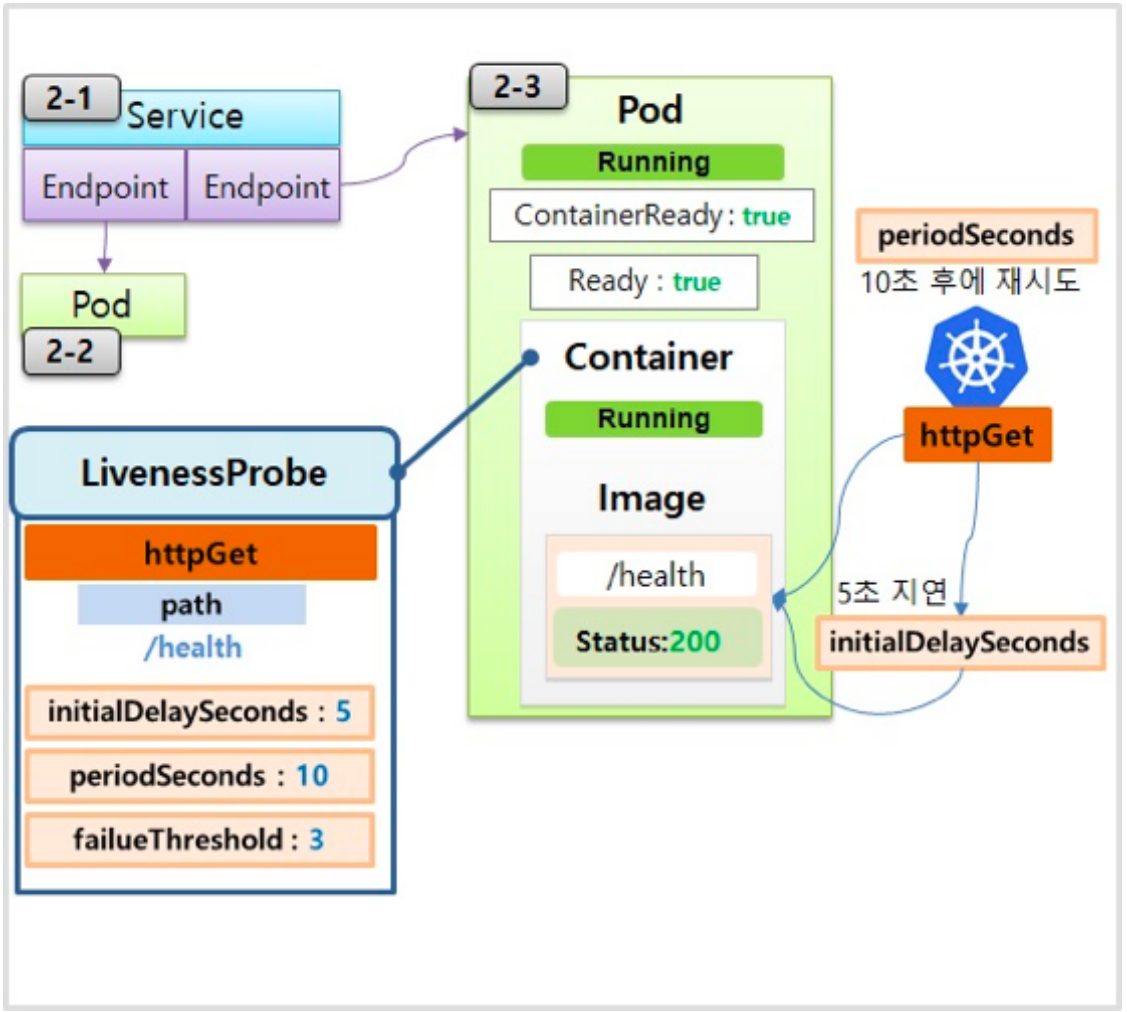


Probe Methods (httpGet, tcpSocket, exec)

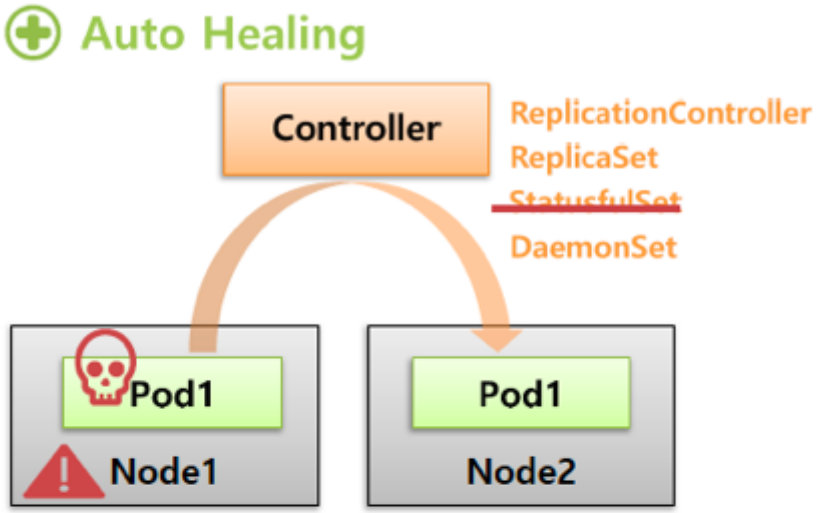


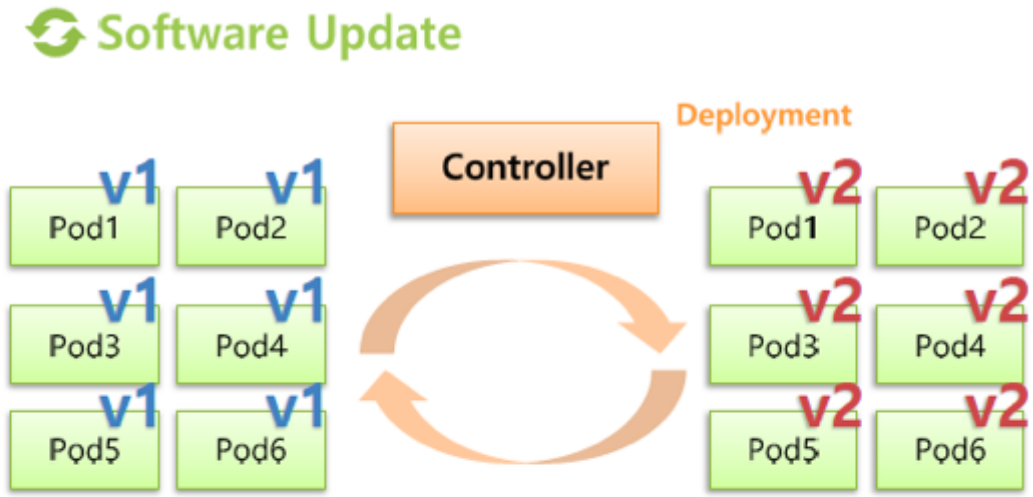
Liveness & Readiness Probe Example

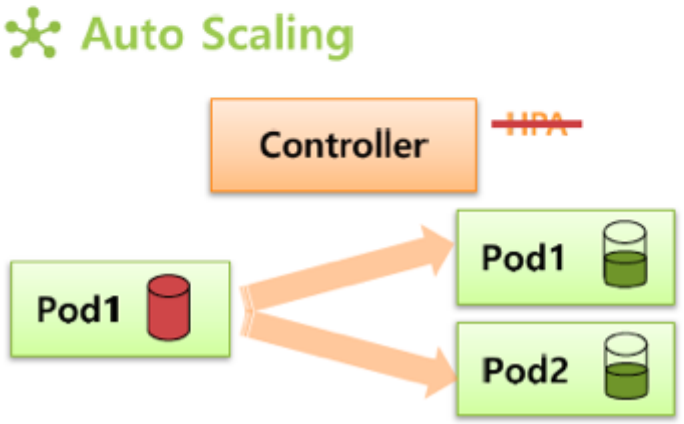


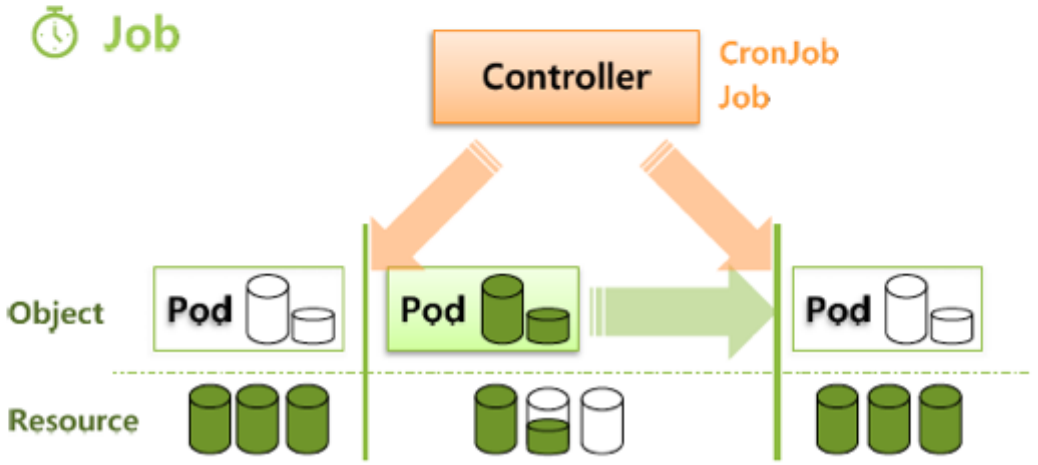


Controllers (in Controller Manager)

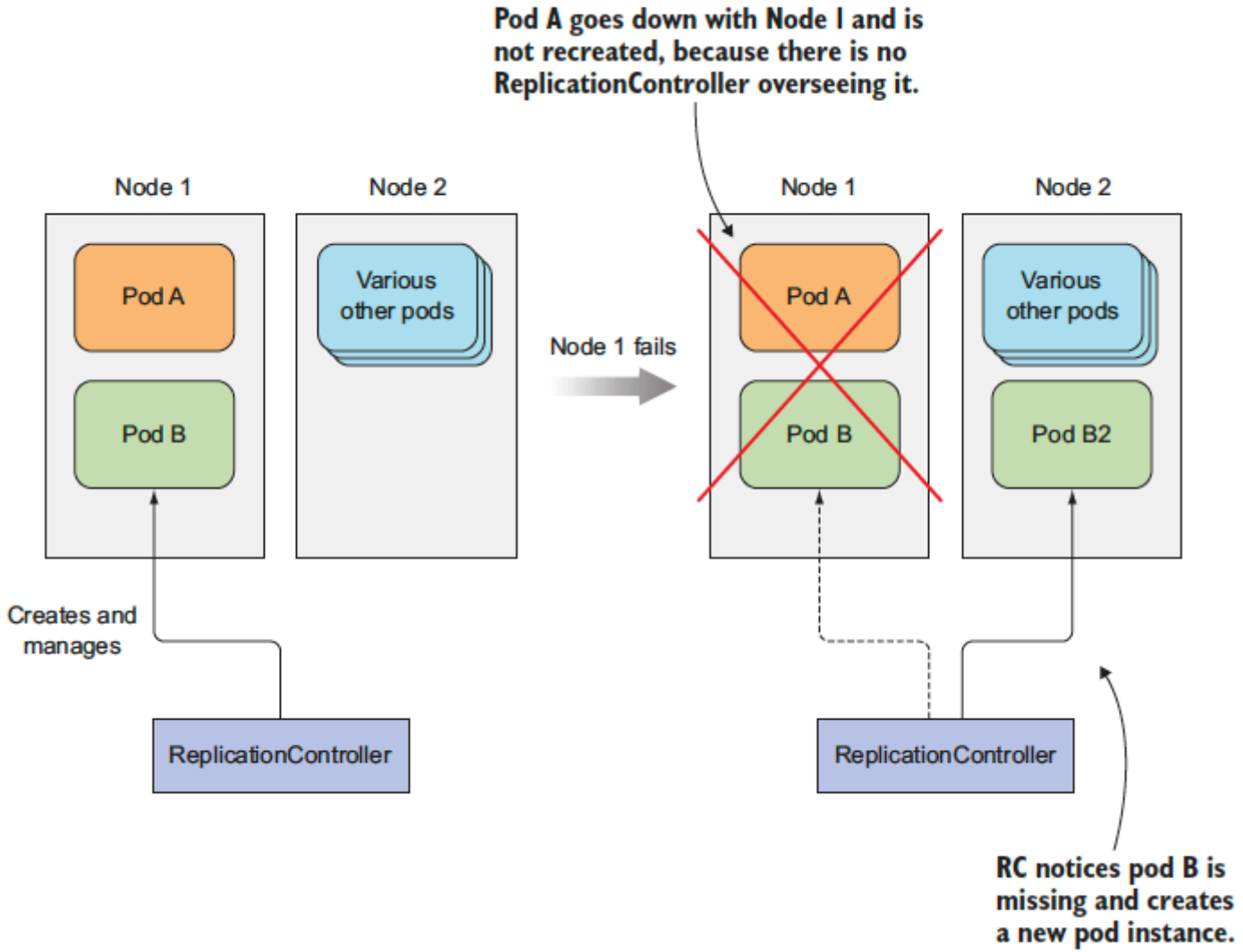




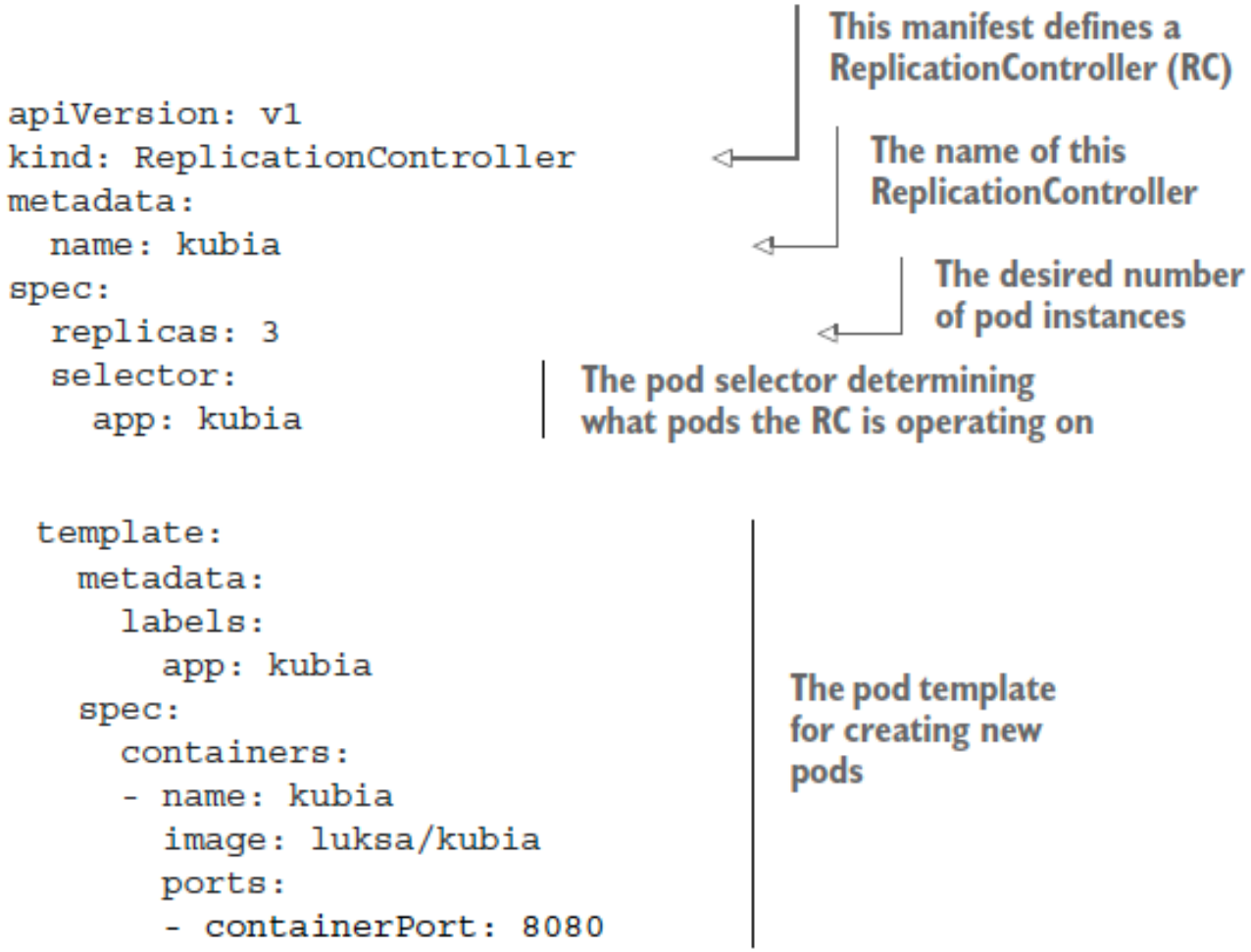




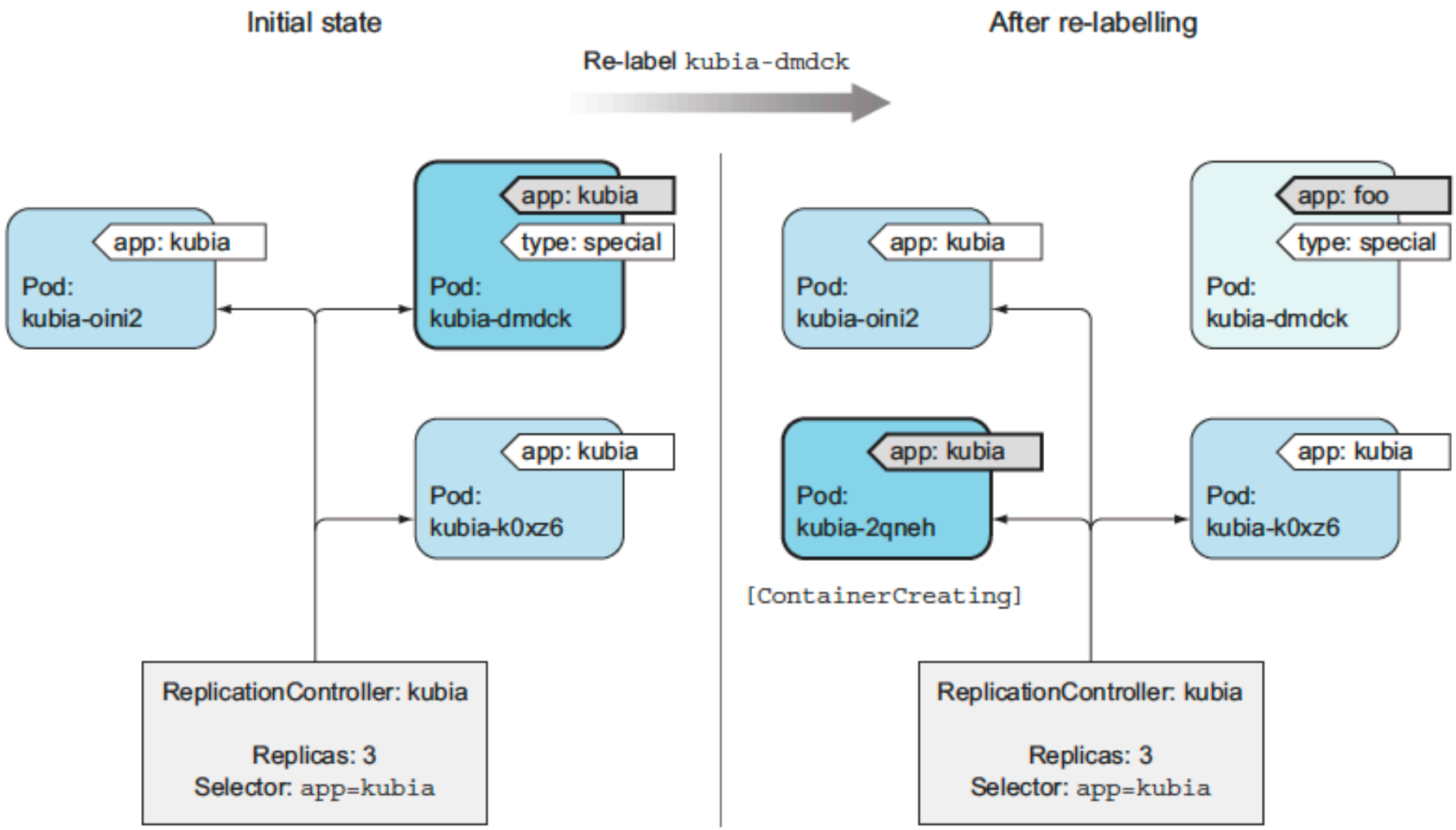
ReplicationController



ReplicationController: YAML File



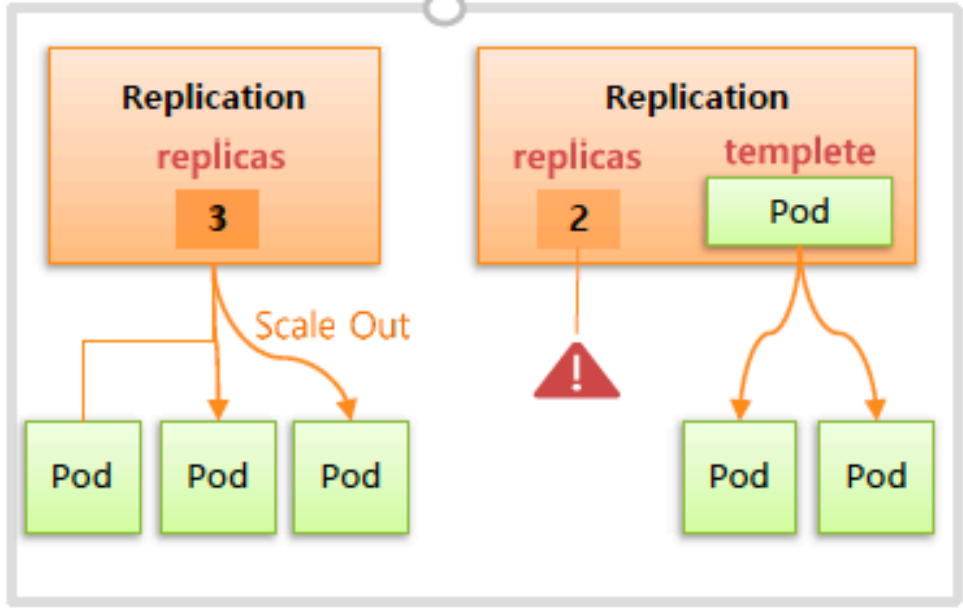
ReplicationController: Re-Labelling

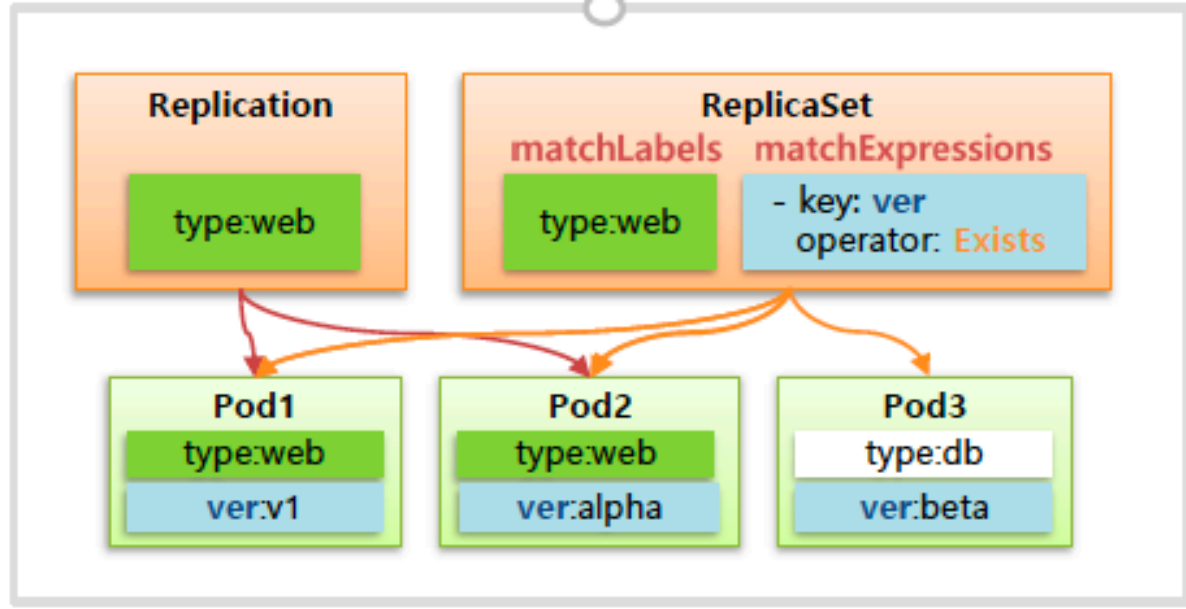


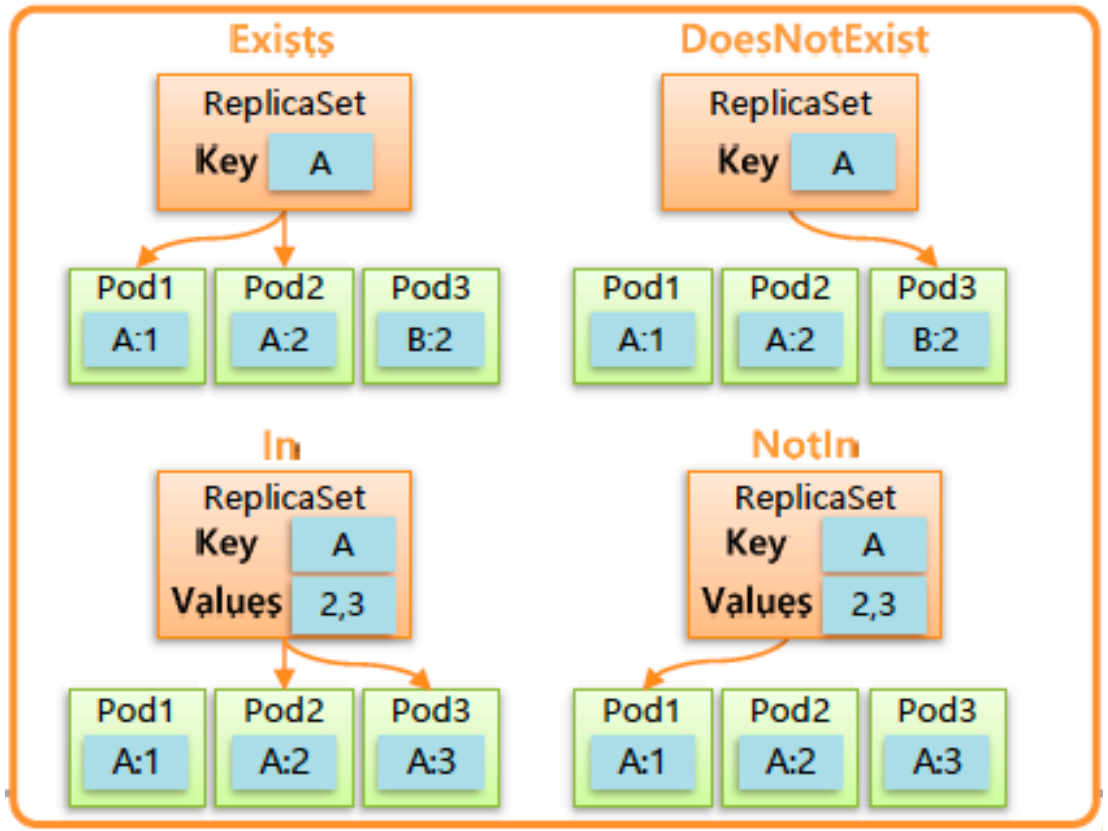
ReplicaSet

- apiVersion: apps/v1beta2
- ReplicaSet is the next-generation Replication Controller

```
selector:
  # ReplicationController는 matchLabels만 사용가능
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
    - {key: service, operator: Exists, values: [user]}
    - {key: service, operator: DoesNotExist, values: [db]}
```

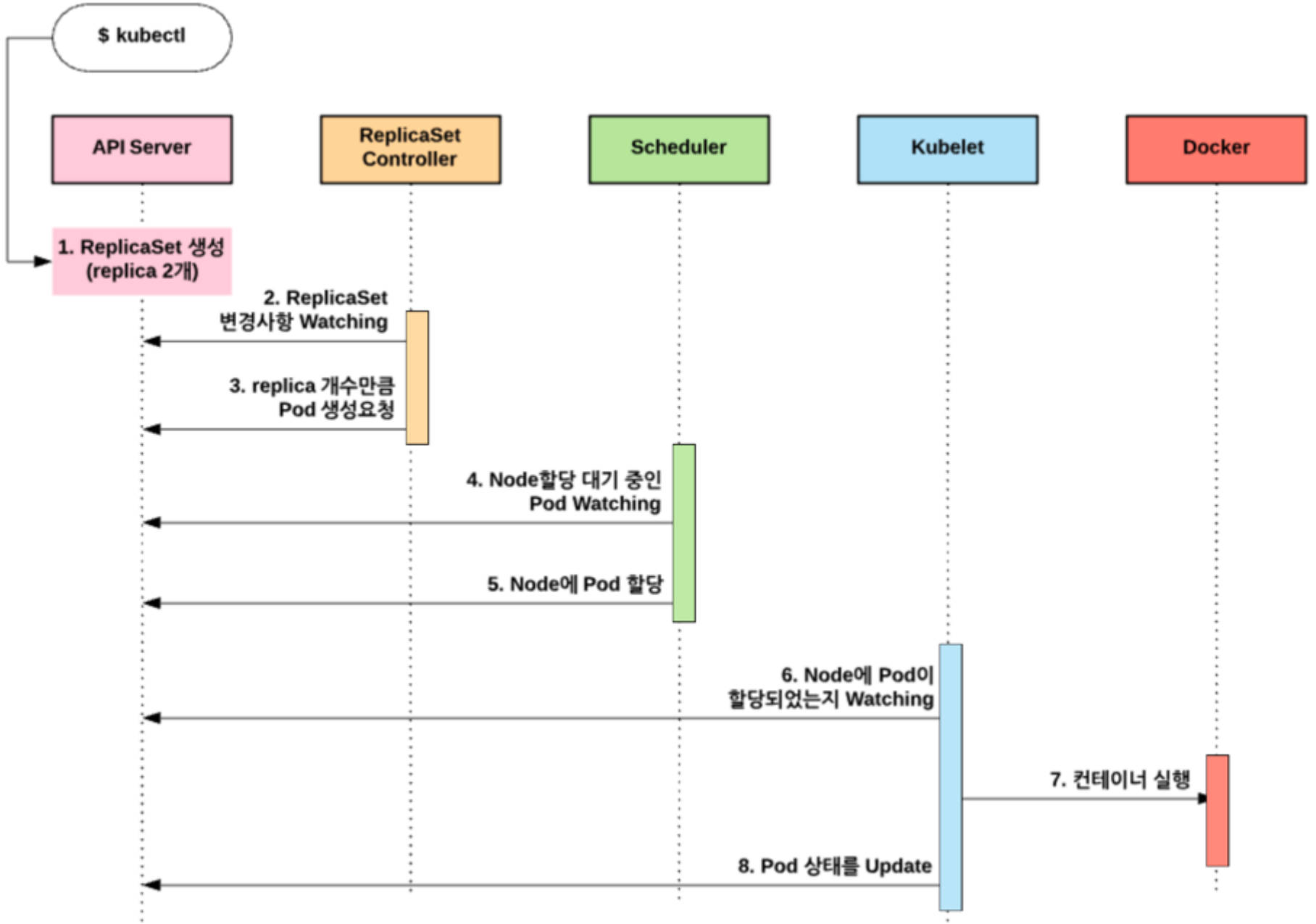






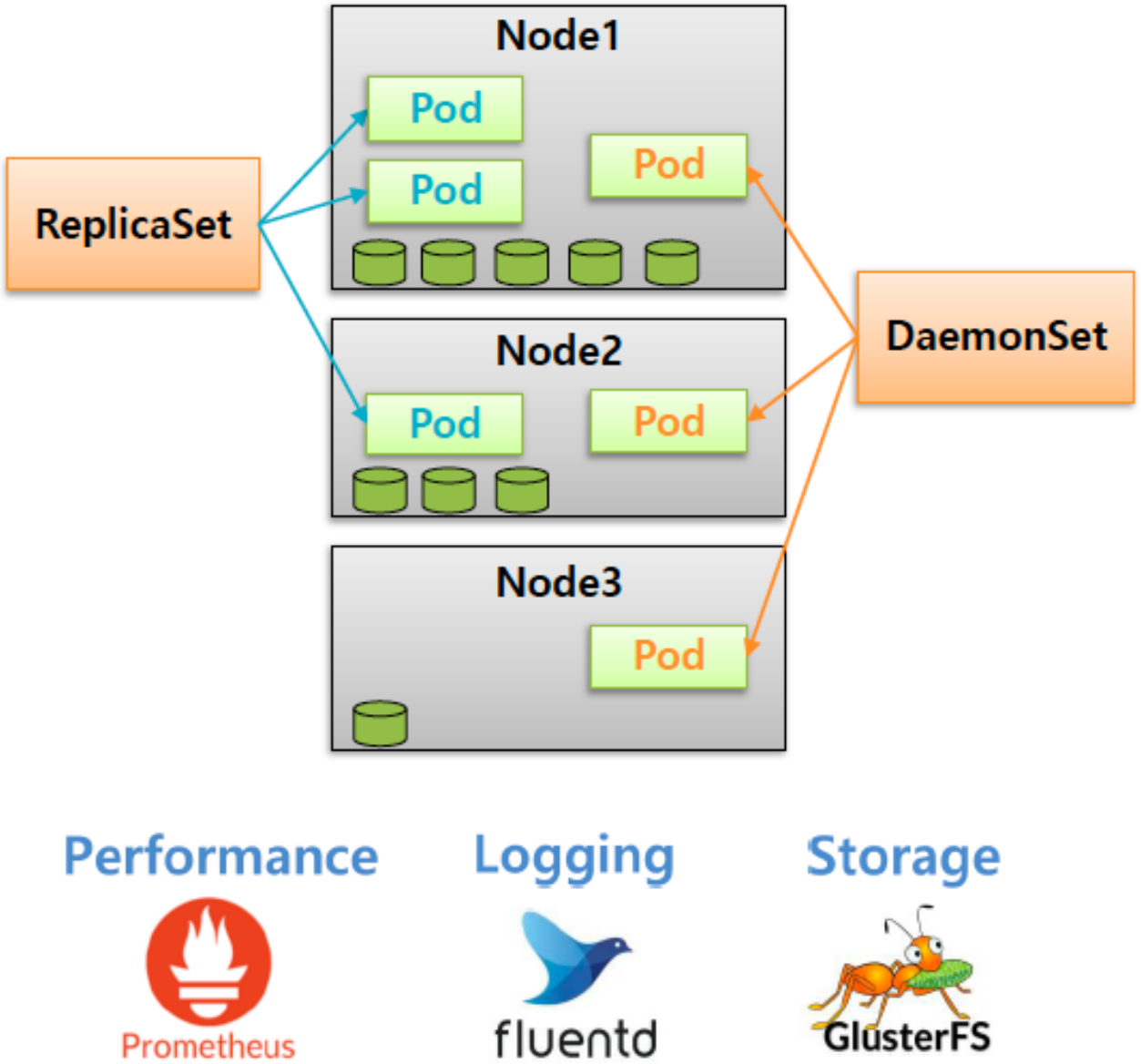

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: replica-1
spec:
 replicas: 3
 selector:
   matchLabels:
     type: web
   matchExpressions:
   - {key: ver, operator: Exists}
 template:
   metadata:
     name: pod
...
```

ReplicaSet

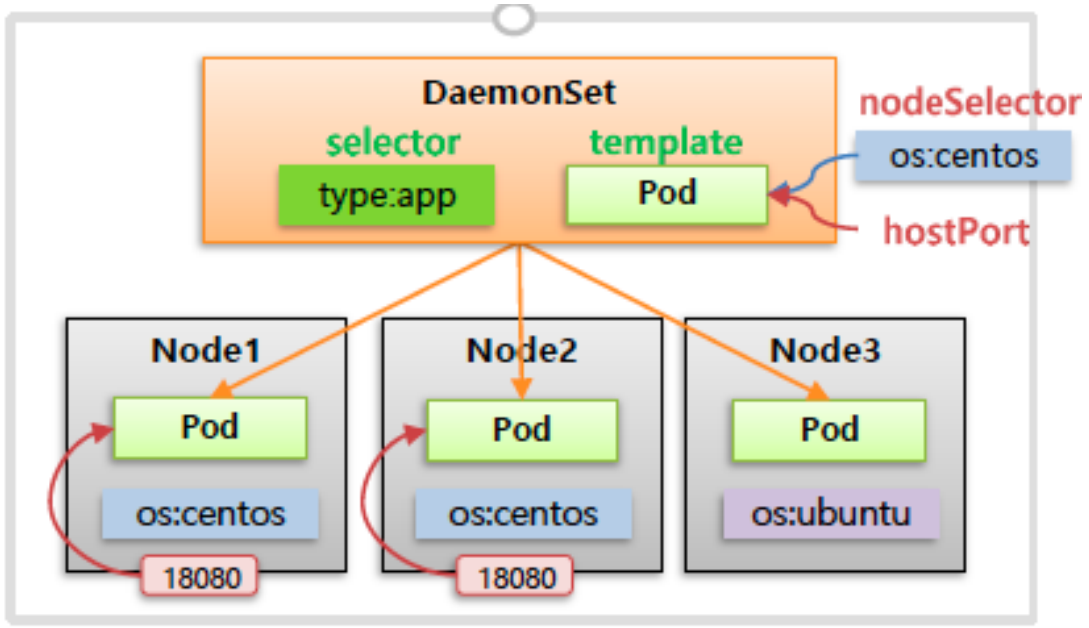

$ kubectl
API Server
ReplicaSet Controller
Scheduler
Kubelet
Docker
1. ReplicaSet 생성 (replica 2개)
2. ReplicaSet 변경사항 Watching
3. replica 개수만큼 Pod 생성요청
4. Node할당 대기 중인 Pod Watching
5. Node에 Pod 할당
6. Node에 Pod이 할당되었는지 Watching
7. 컨테이너 실행
8. Pod 상태를 Update

DaemonSet

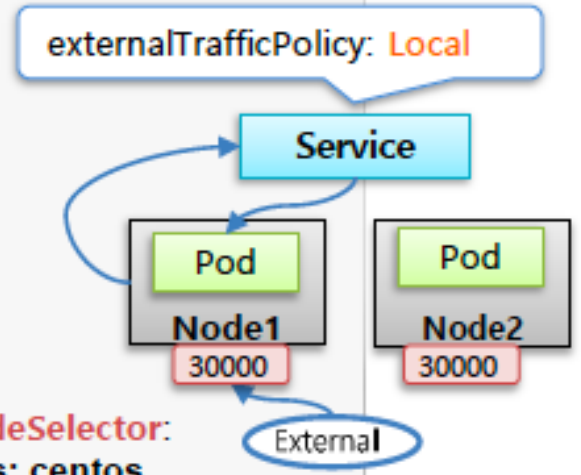
- apiVersion: apps/v1beta2



DaemonSet: YAML File

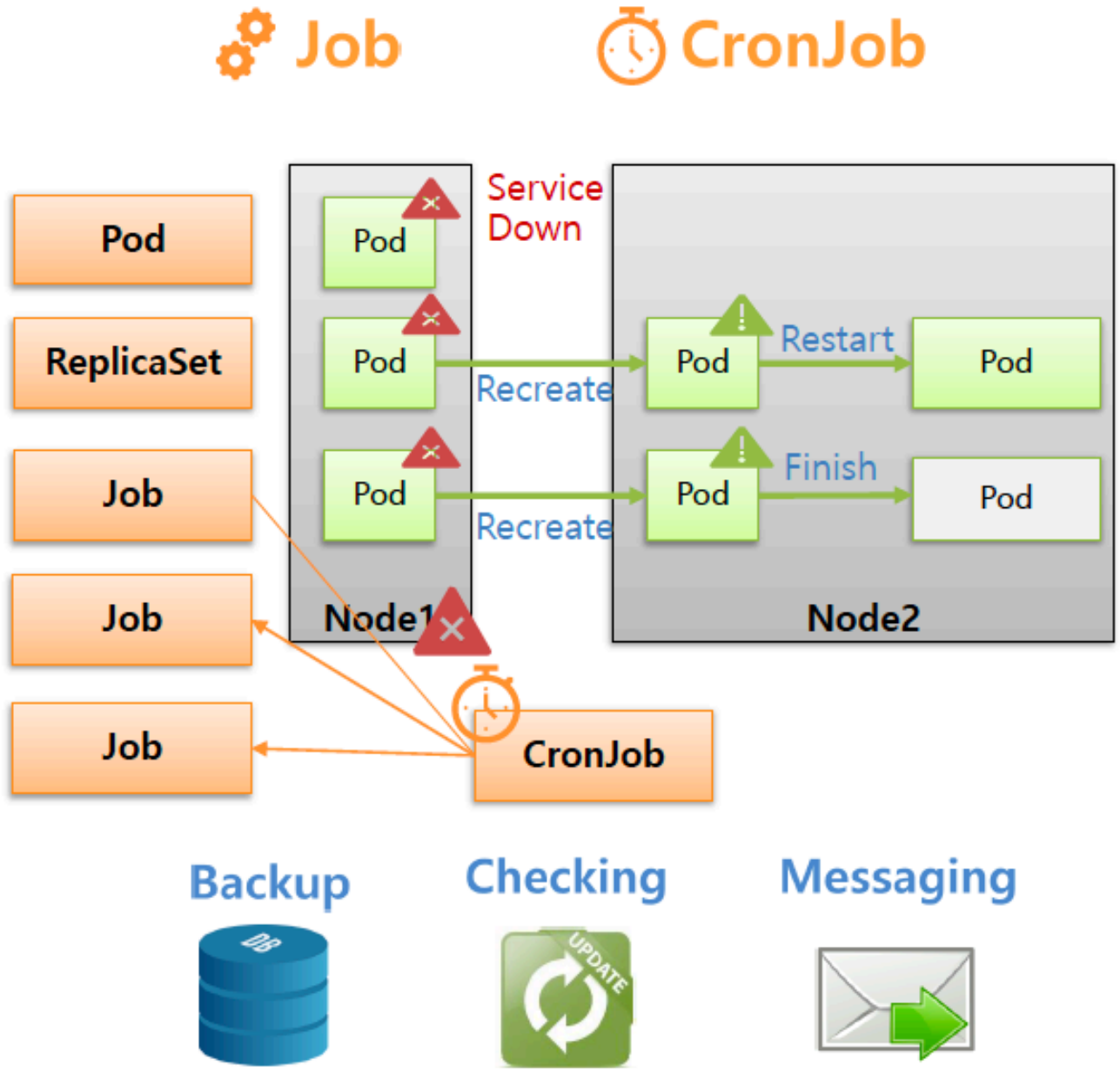



```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: daemonset-1
spec:
 selector:
   matchLabels:
     type: app
 template:
   metadata:
     labels:
       type: app
   spec:
     nodeSelector:
       os: centos
     containers:
     - name: container
       image: tmkube/app
       ports:
       - containerPort: 8080
         hostPort: 18080
```

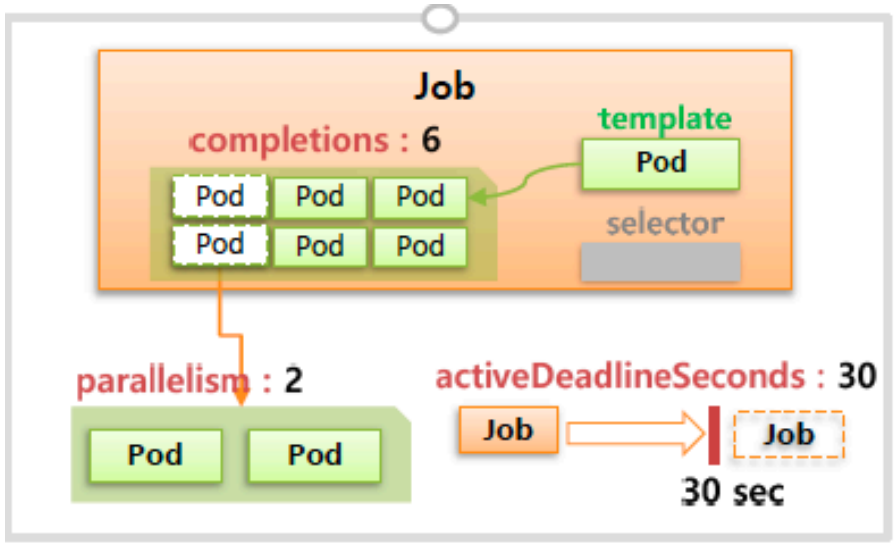


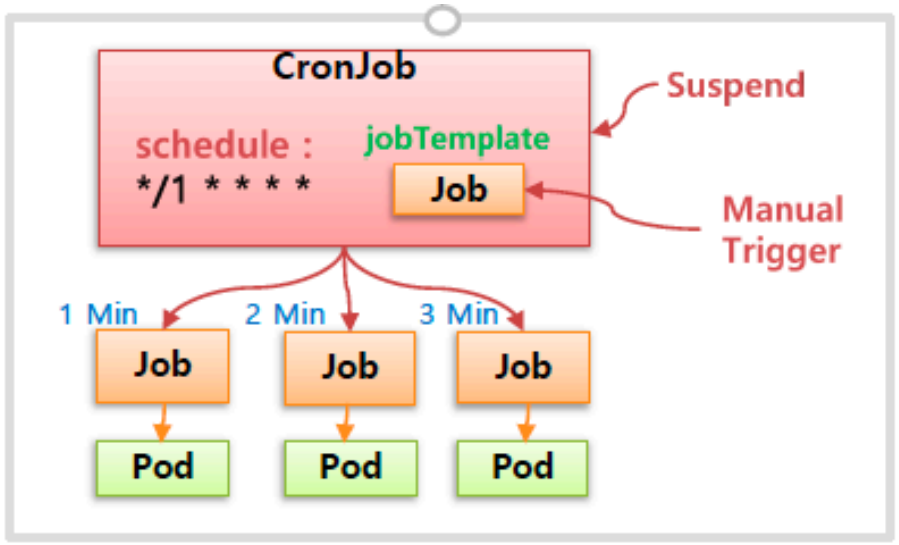
Job, CronJob

- apiVersion: batch/v1



Job, CronJob: YAML File

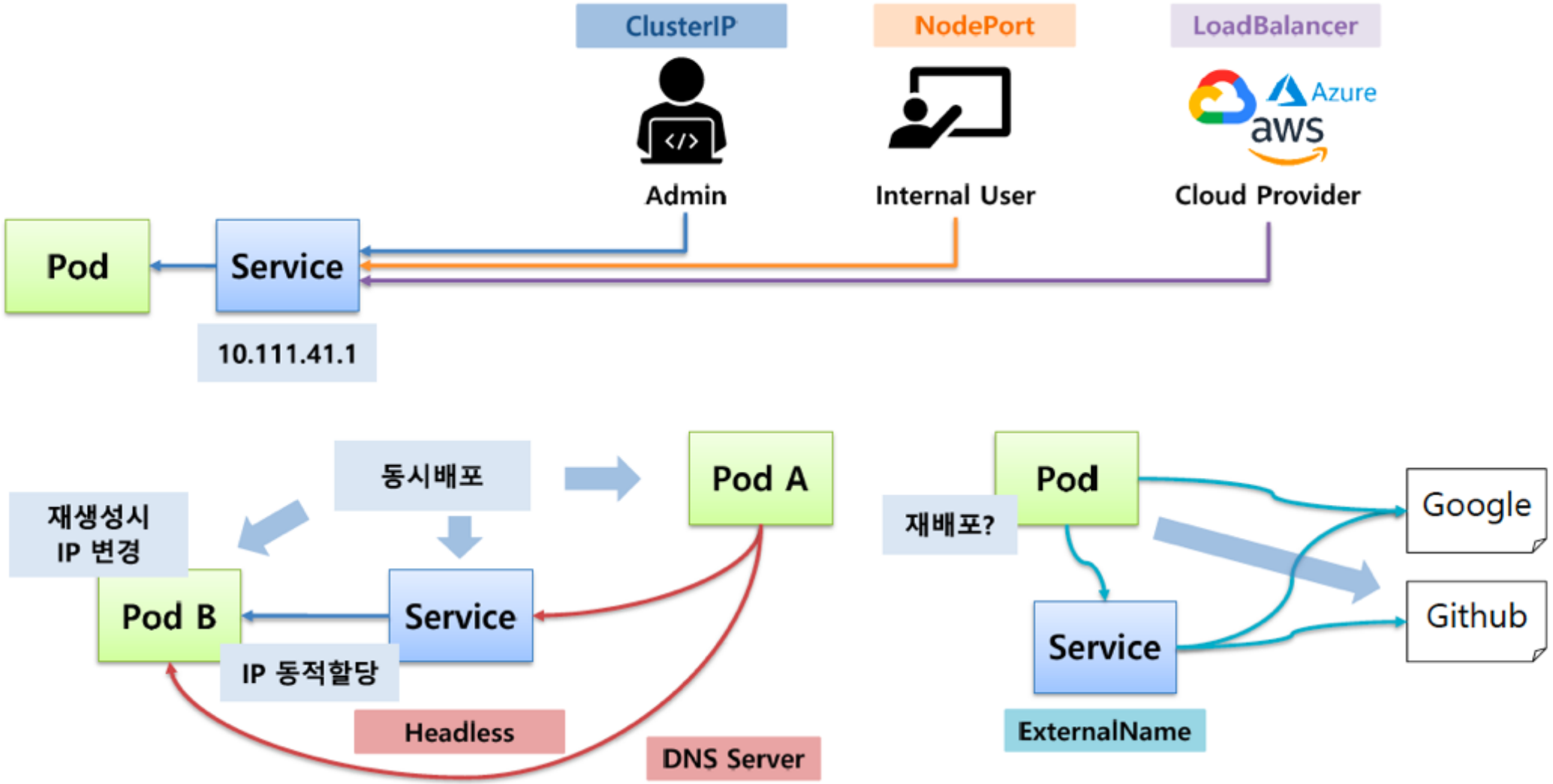


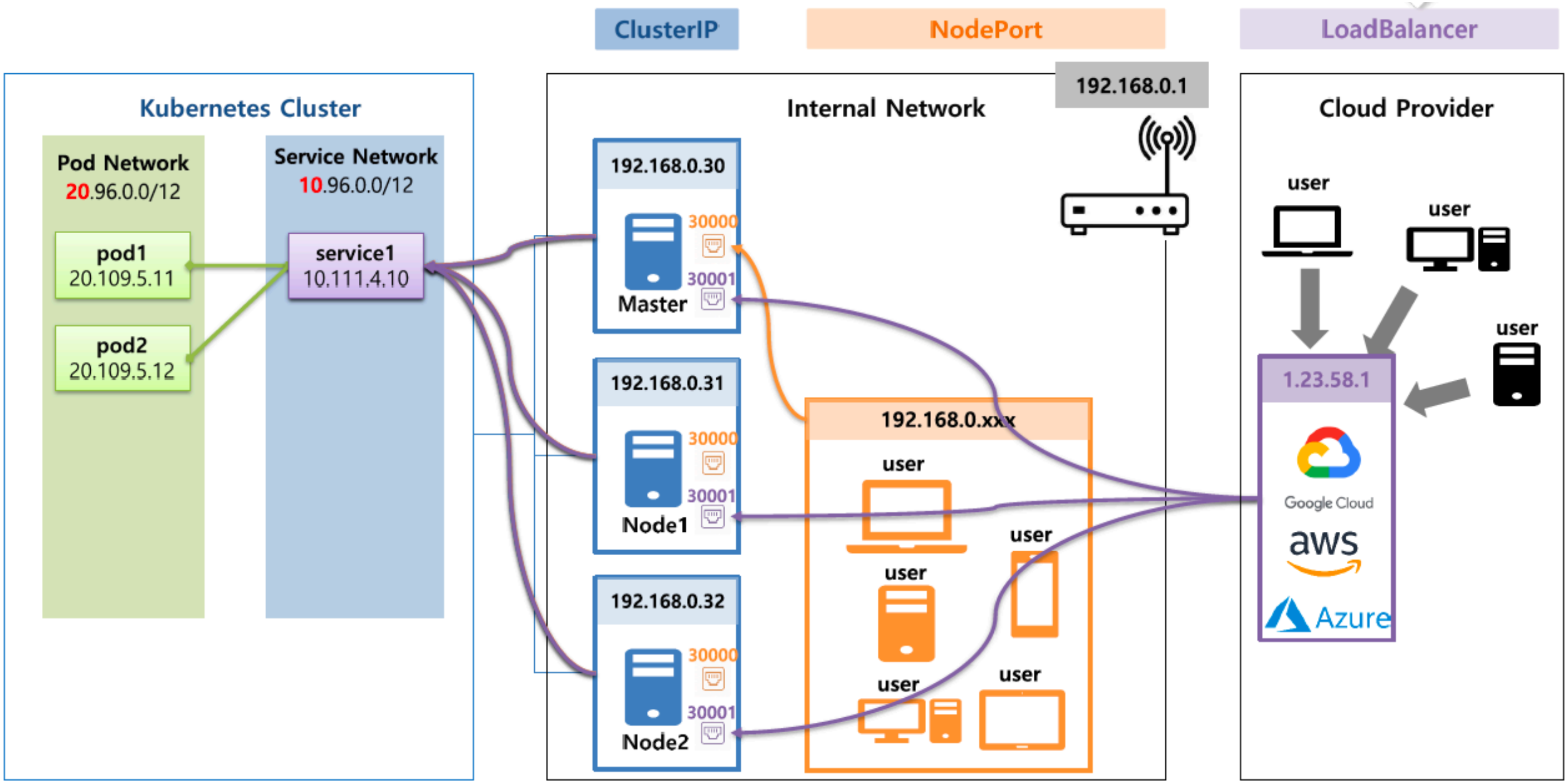



```yaml
apiVersion: batch/v1
kind: Job
metadata:
 name: job-1
spec:
  completions: 6
  parallelism: 2
  activeDeadlineSeconds: 30
  template:
   spec:
     restartPolicy: Never
     containers:
     - name: container
       image: tmkube/init
```

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: cron-job
spec:
  schedule: "*/1 * * * *"
  concurrencyPolicy: Allow
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
          - name: container
            image: tmkube/app
```

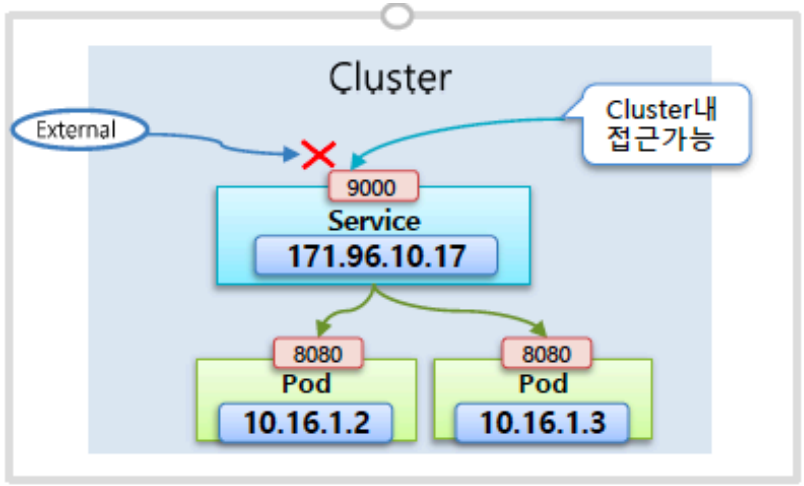
Service: ClusterIP, NodePort, LoadBalancer, ExternalName, Headless




ClusterIP
NodePort
LoadBalancer
Kubernetes Cluster
Pod Network
20.96.0.0/12
pod1
20.109.5.11
pod2
20.109.5.12
Service Network
10.96.0.0/12
service1
10.111.4.10
Internal Network
192.168.0.1
192.168.0.30
30000
30001
Master
192.168.0.31
30000
30001
Node1
192.168.0.32
30000
30001
Node2
192.168.0.xxx
user
user
user
user
user
Cloud Provider
user
user
user
1.23.58.1
Google Cloud
aws
Azure

Service: ClusterIP, NodePort, LoadBalancer

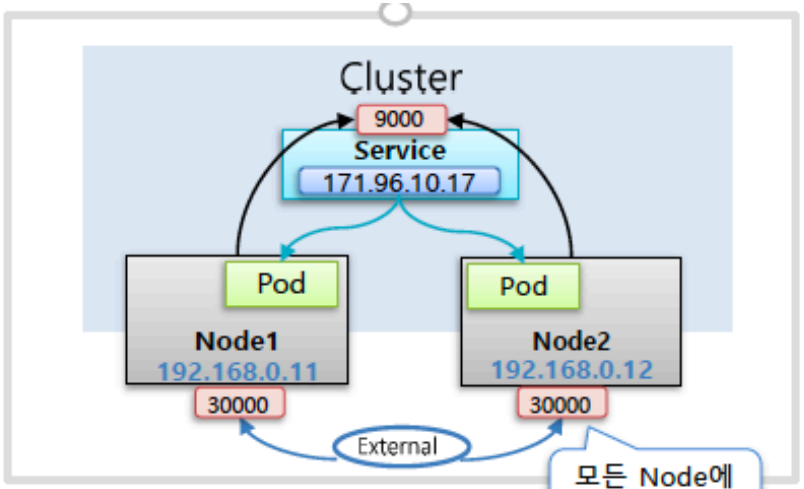
- 서비스 디버깅
- 내부 트래픽/대쉬보드
- 인증된 사용자 연결




```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-1
spec:
  selector:
    app: pod
  ports:
    - port: 9000
      targetPort: 8080
  type: ClusterIP
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  labels:
    app: pod
spec:
  containers:
  - name: container
    image: tmkube/app
    ports:
    - containerPort: 8080
```

- 데모나 임시 연결용
- 프로덕션 환경(X)

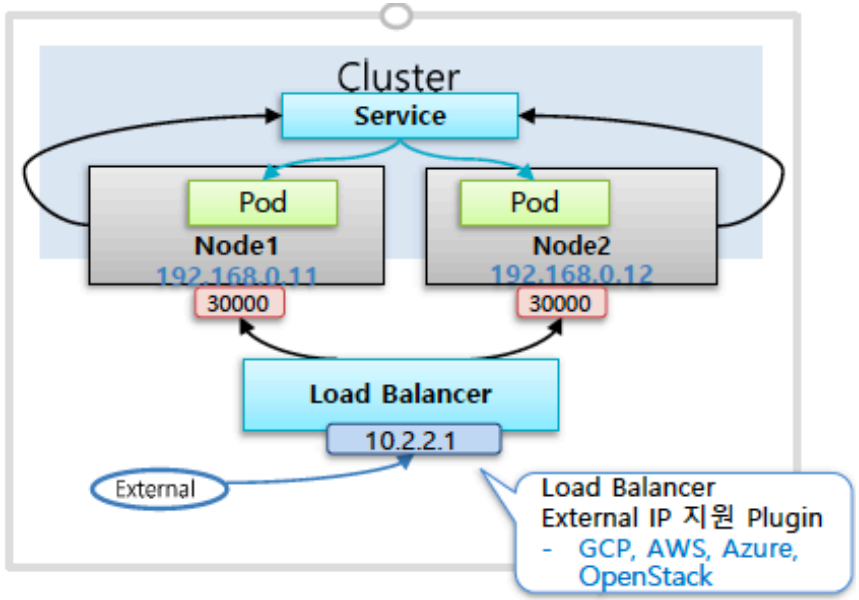



```yaml
apiVersion: v1
kind: Service
metadata:
 name: svc-2
spec:
 selector:
    app: pod
 ports:
   - port: 9000
     targetPort: 8080
     nodePort: 30000
 type: NodePort
```

30000~32767

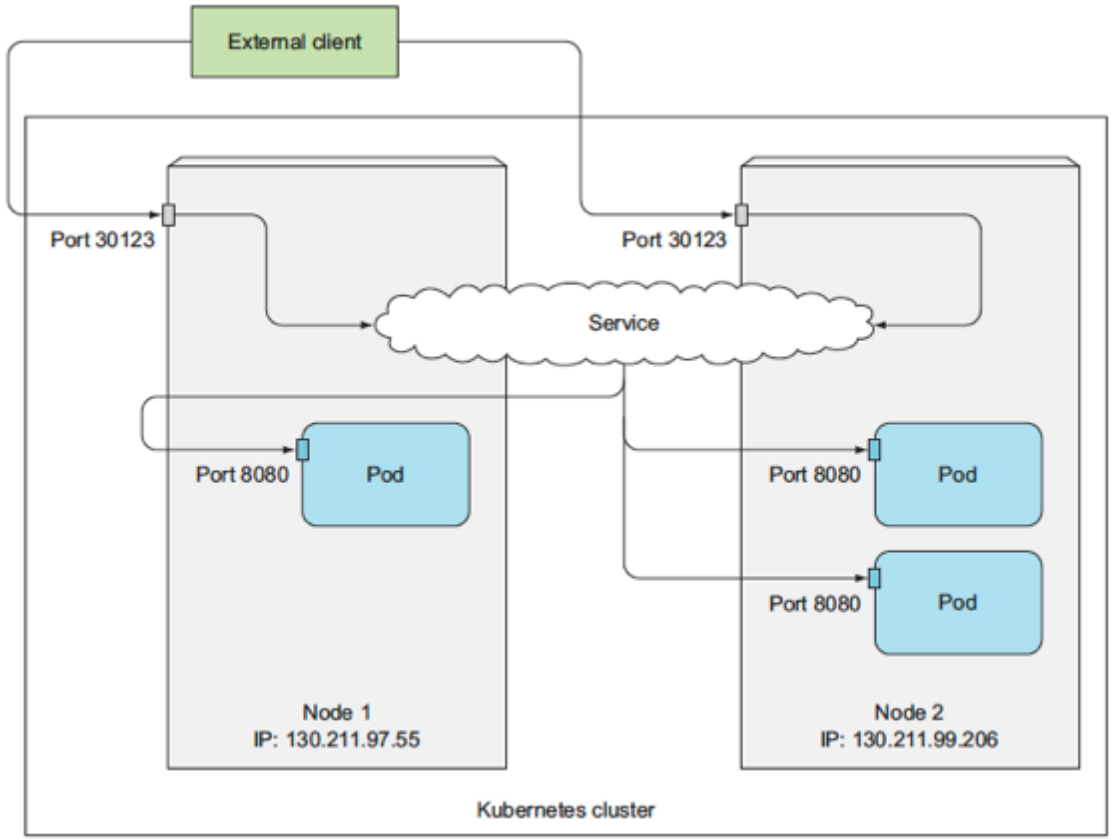
externalTrafficPolicy: Local

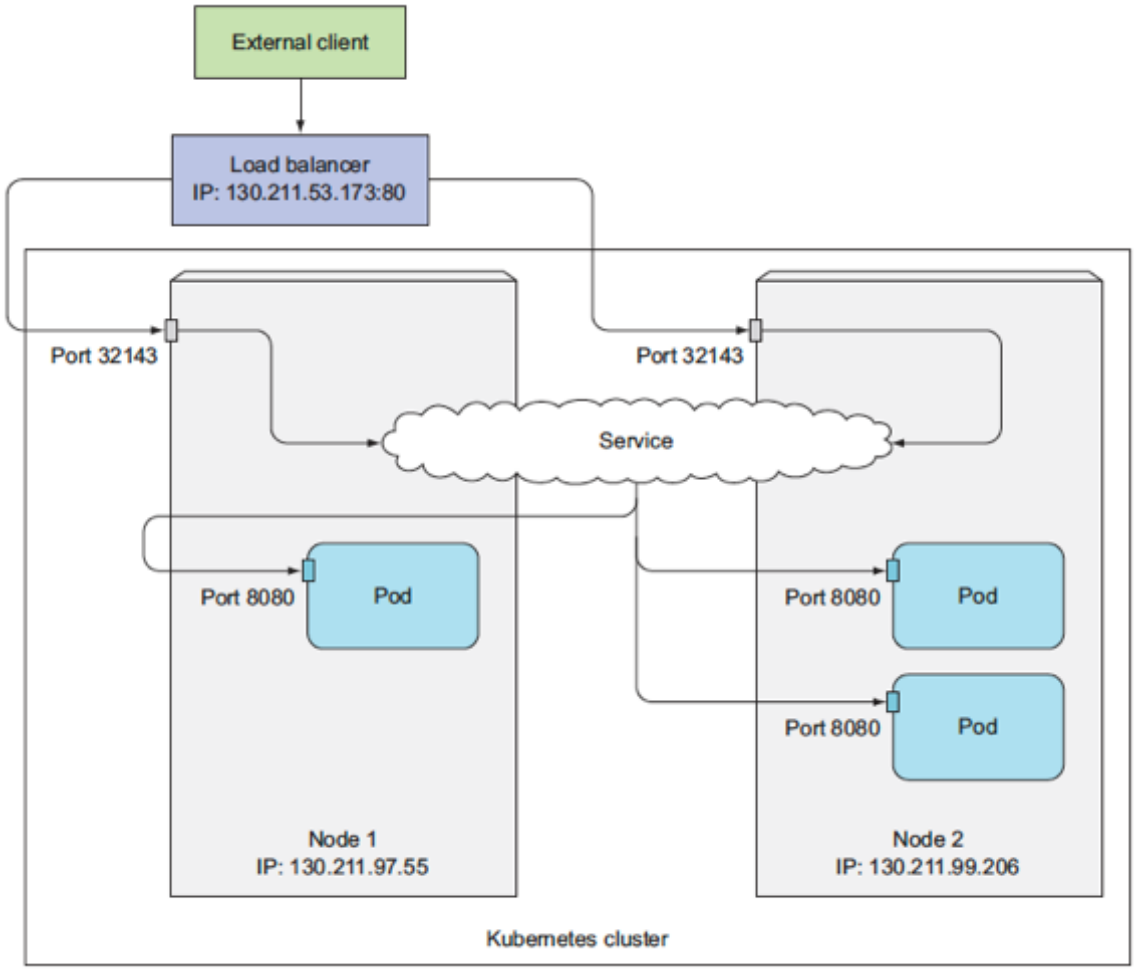
- 외부 시스템 노출용



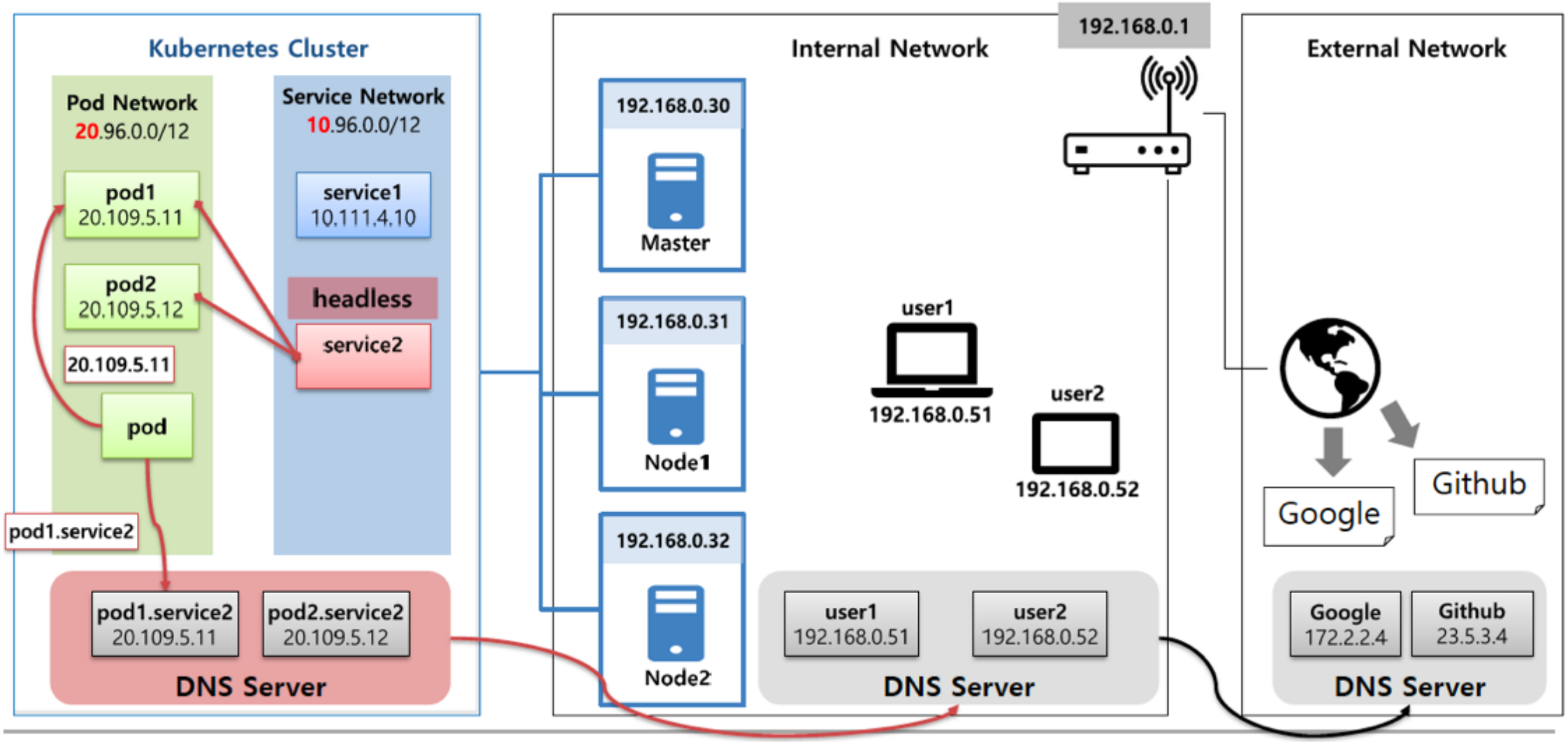

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-3
spec:
  selector:
    app: pod
  ports:
    - port: 9000
      targetPort: 8080
  type: LoadBalancer
```

NodePort vs LoadBalancer

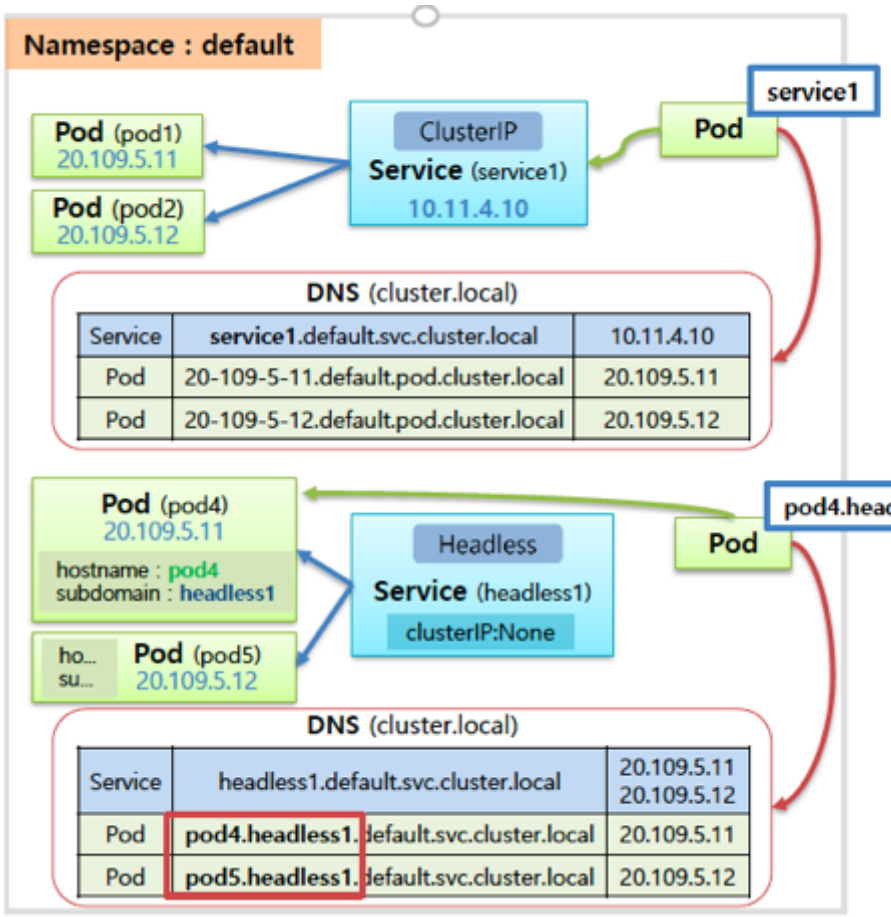




Headless (1)


Kubernetes Cluster
Pod Network
20.96.0.0/12
Service Network
10.96.0.0/12
pod1
20.109.5.11
service1
10.111.4.10
pod2
20.109.5.12
headless
service2
20.109.5.11
pod
pod1.service2
pod1.service2
20.109.5.11
pod2.service2
20.109.5.12
DNS Server
Internal Network
192.168.0.1
192.168.0.30
Master
192.168.0.31
Node1
192.168.0.32
Node2
user1
192.168.0.51
user2
192.168.0.52
user1
192.168.0.51
user2
192.168.0.52
DNS Server
External Network
Google
Github
Google
172.2.2.4
Github
23.5.3.4
DNS Server

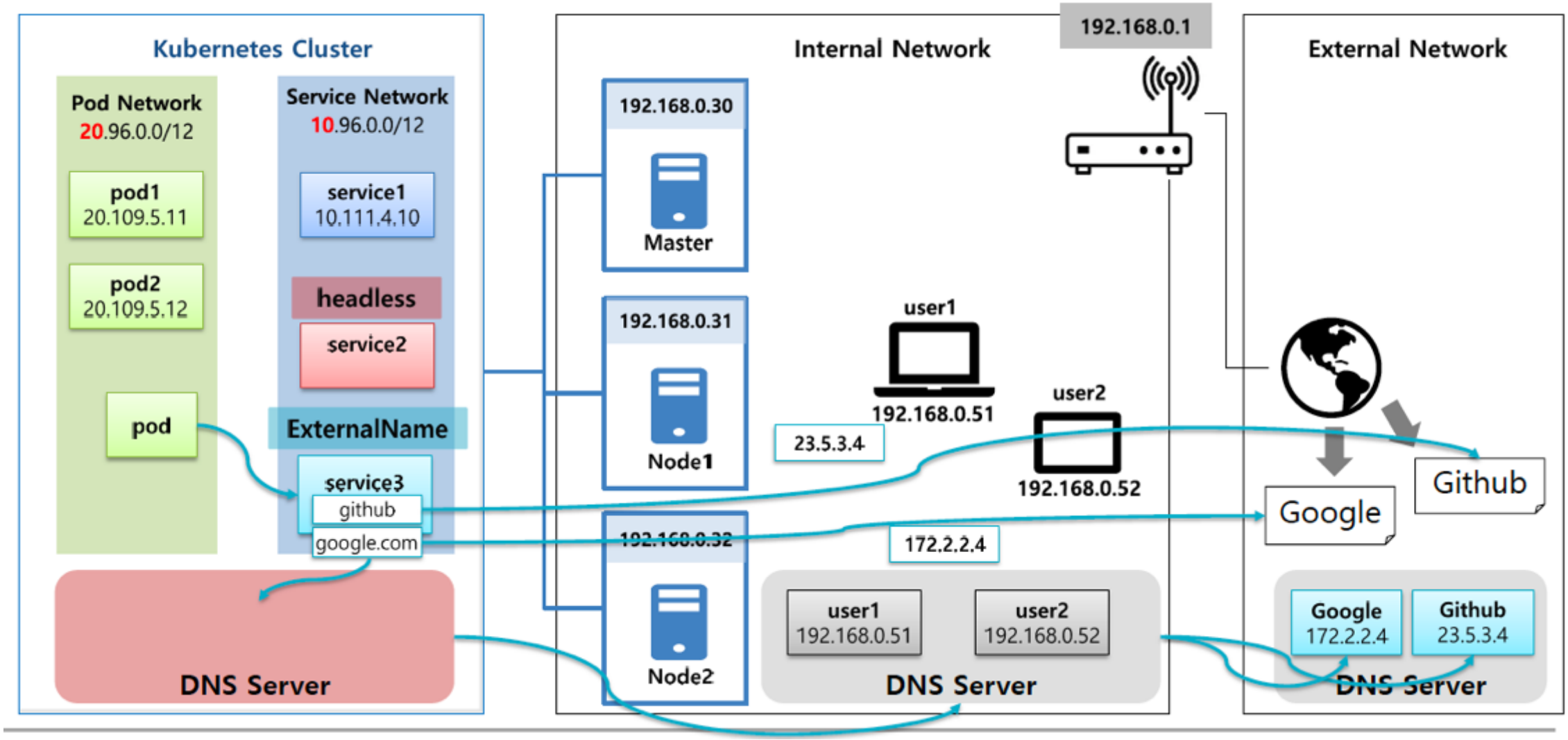
Headless (2)



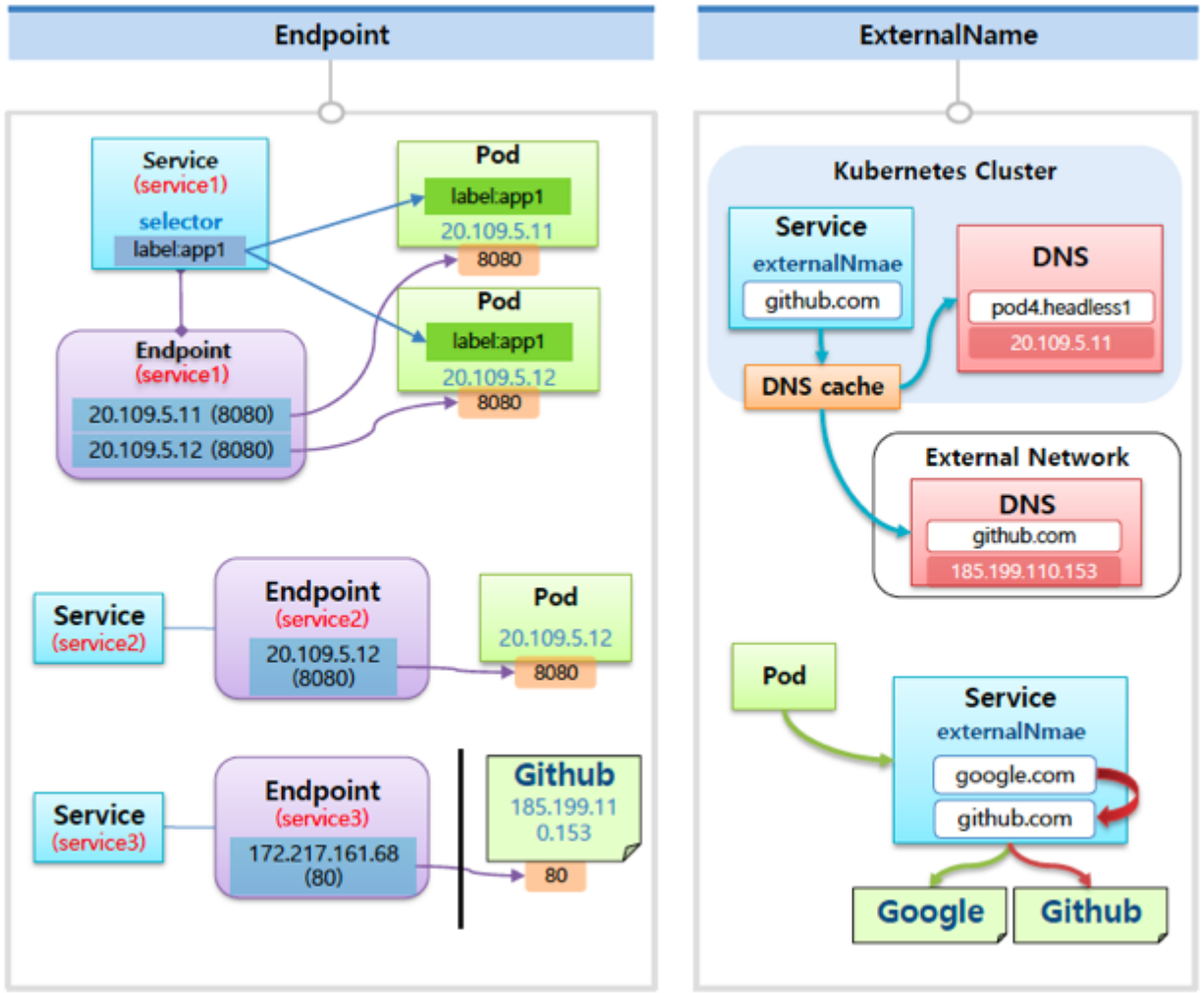
Service	service1.default.svc.cluster.local	10.11.4.10
Pod	20-109-5-11.default.pod.cluster.local	20.109.5.11

FQDN = Fully Qualified Domain Name

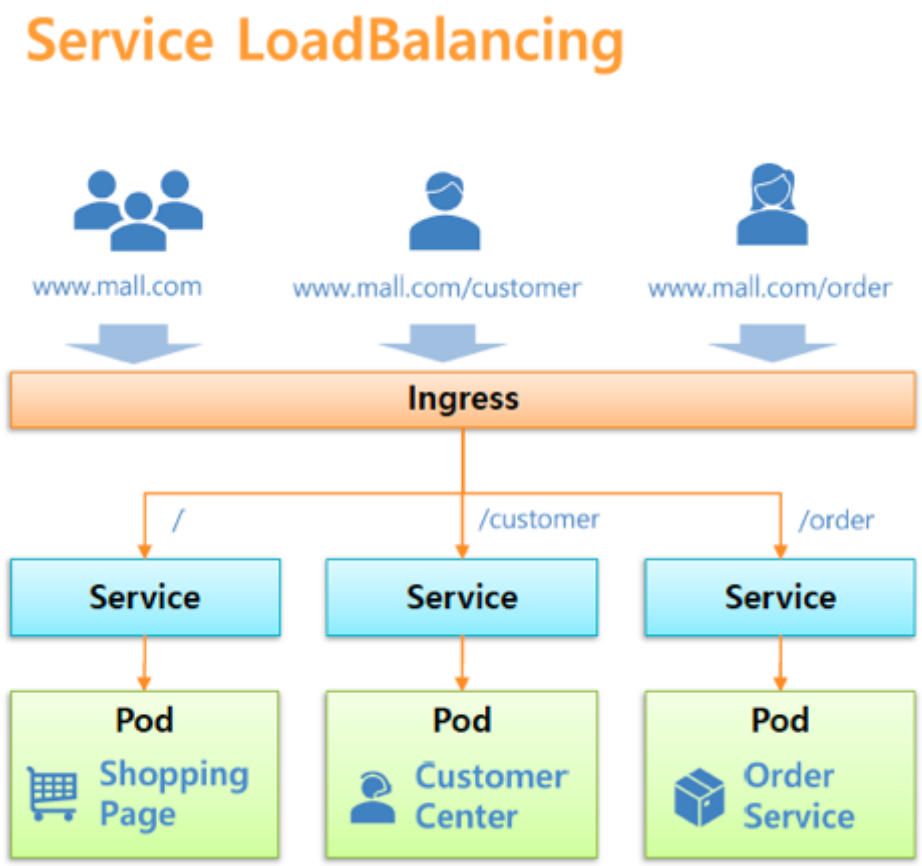
ExternalName

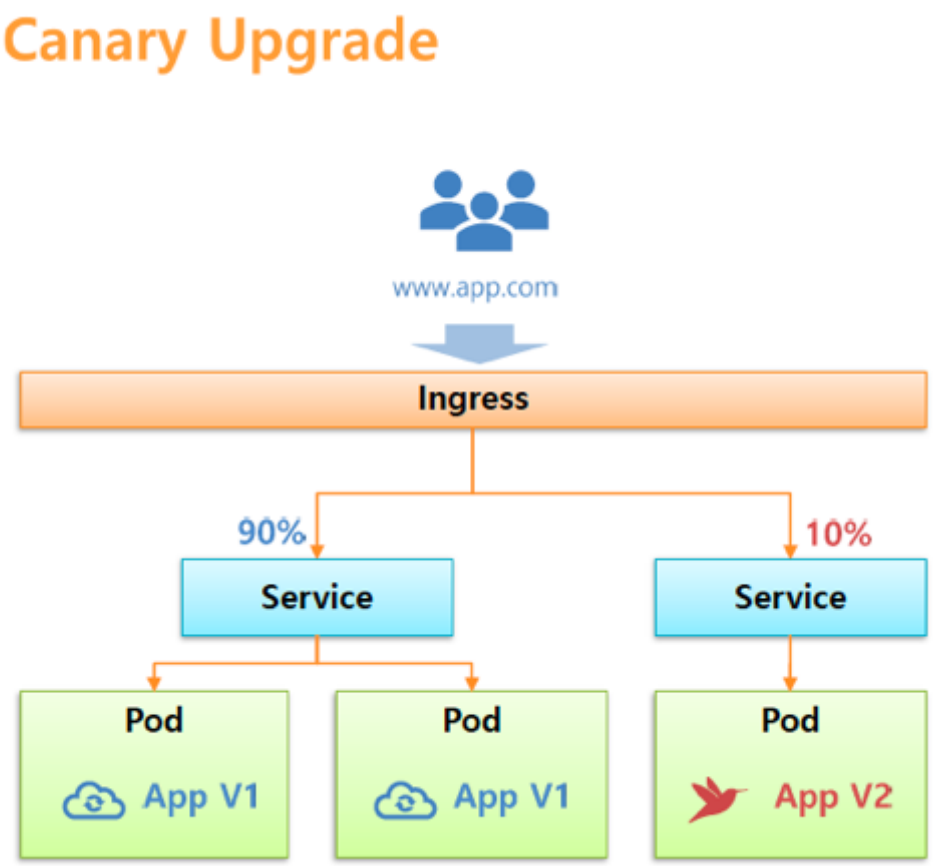


ExternalName & Endpoint

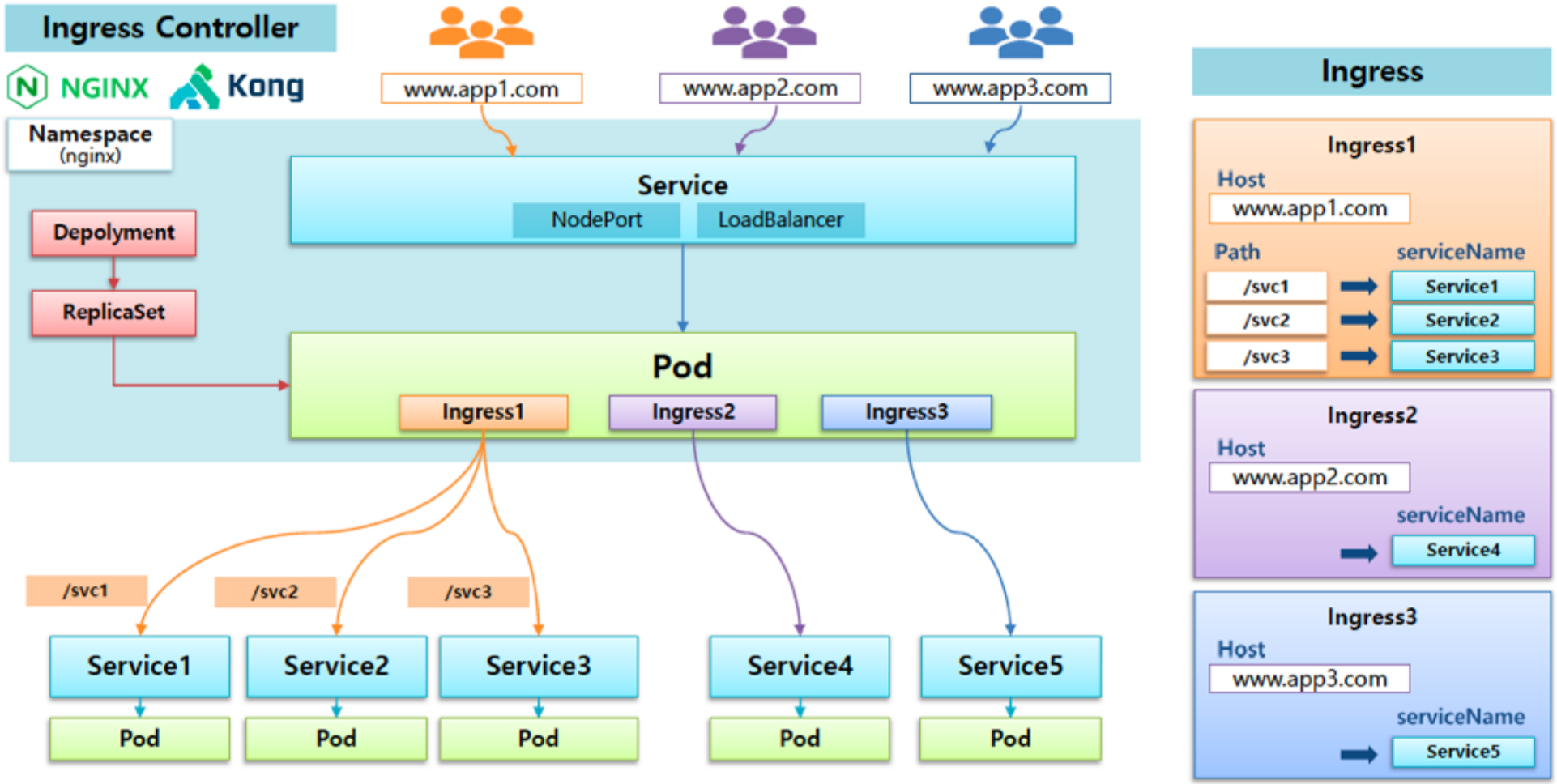


Ingress

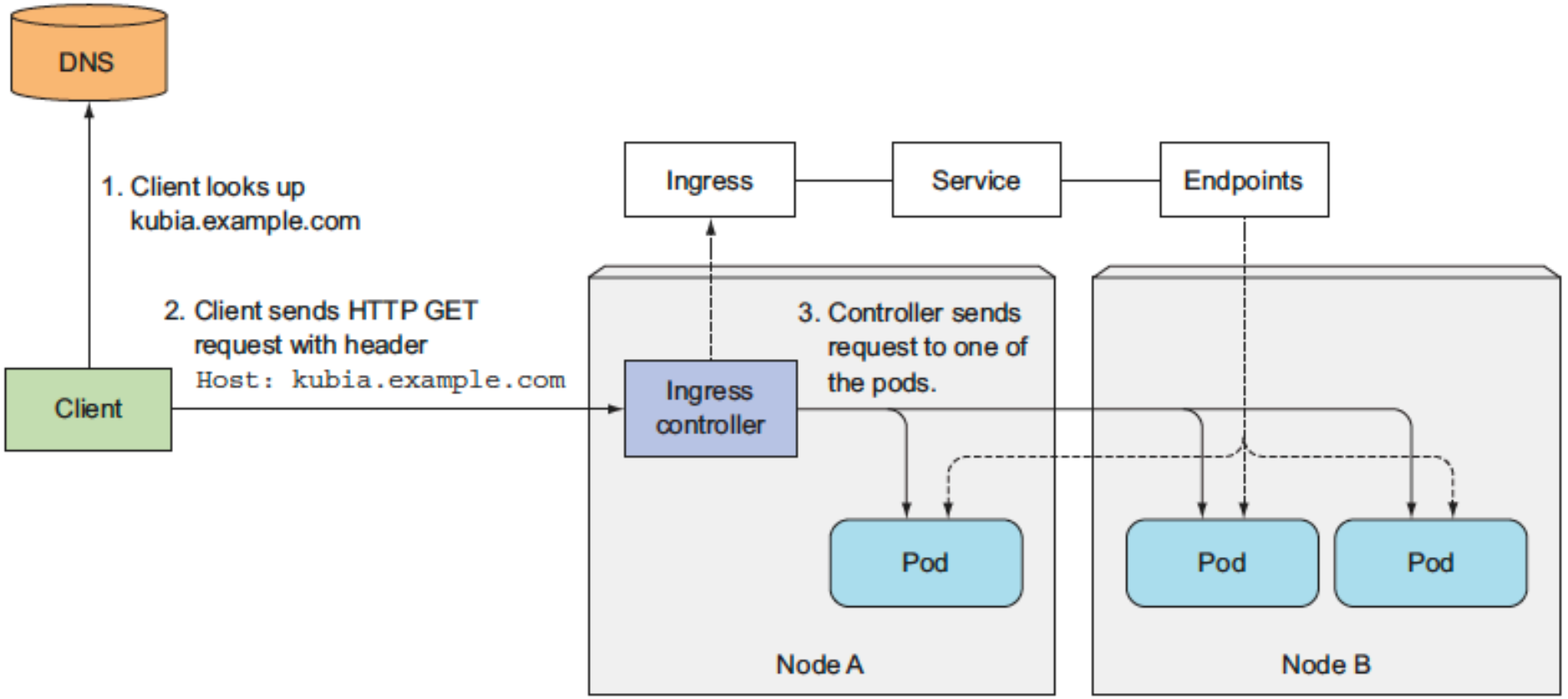

Service LoadBalancing
www.mall.com
www.mall.com/customer
www.mall.com/order
Ingress
/
/customer
/order
Service
Service
Service
Pod
Shopping Page
Pod
Customer Center
Pod
Order Service


Canary Upgrade
www.app.com
Ingress
90%
10%
Service
Service
Pod
App V1
Pod
App V1
Pod
App V2

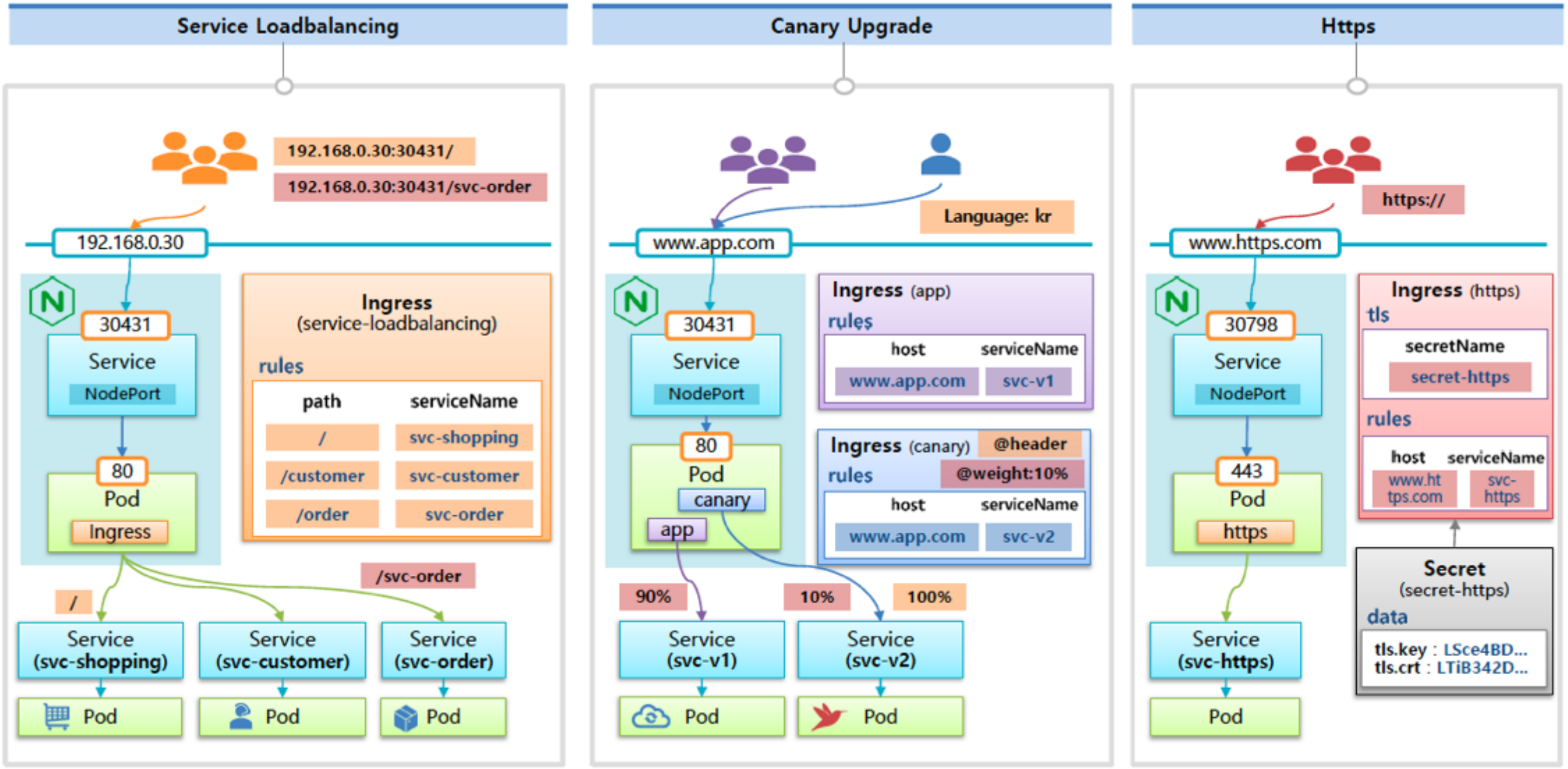
Ingress Controller



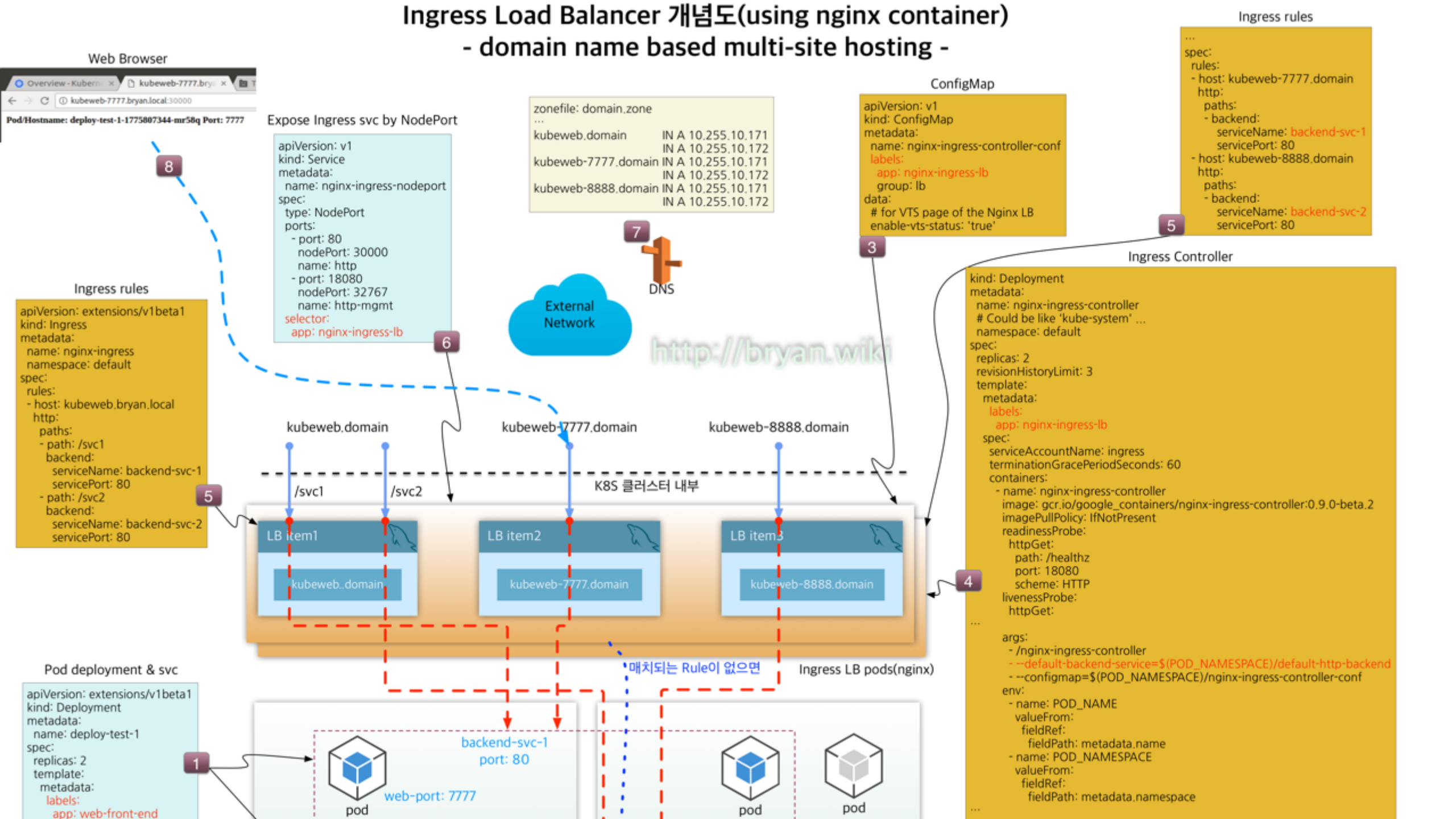
Ingress, Ingress Controller, Endpoints



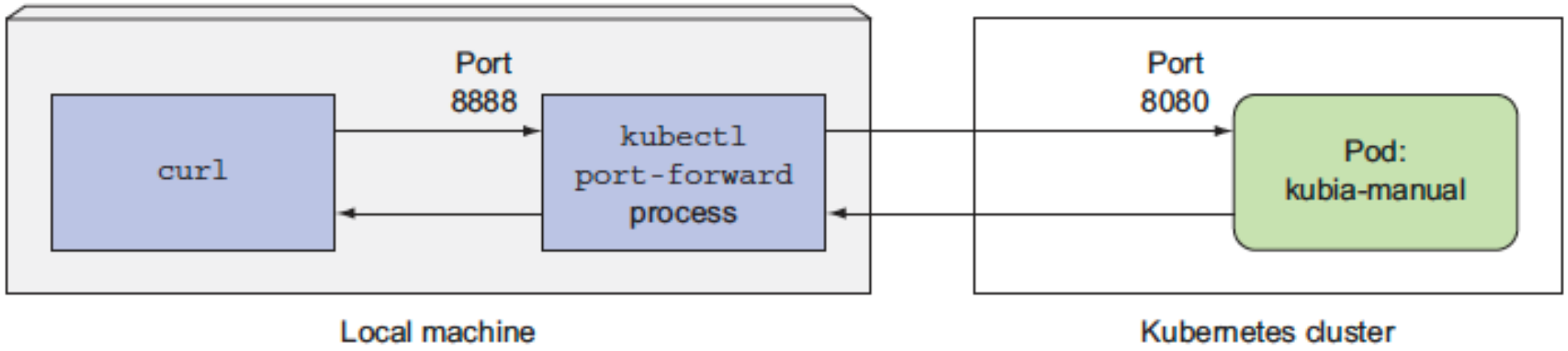
Ingress Examples



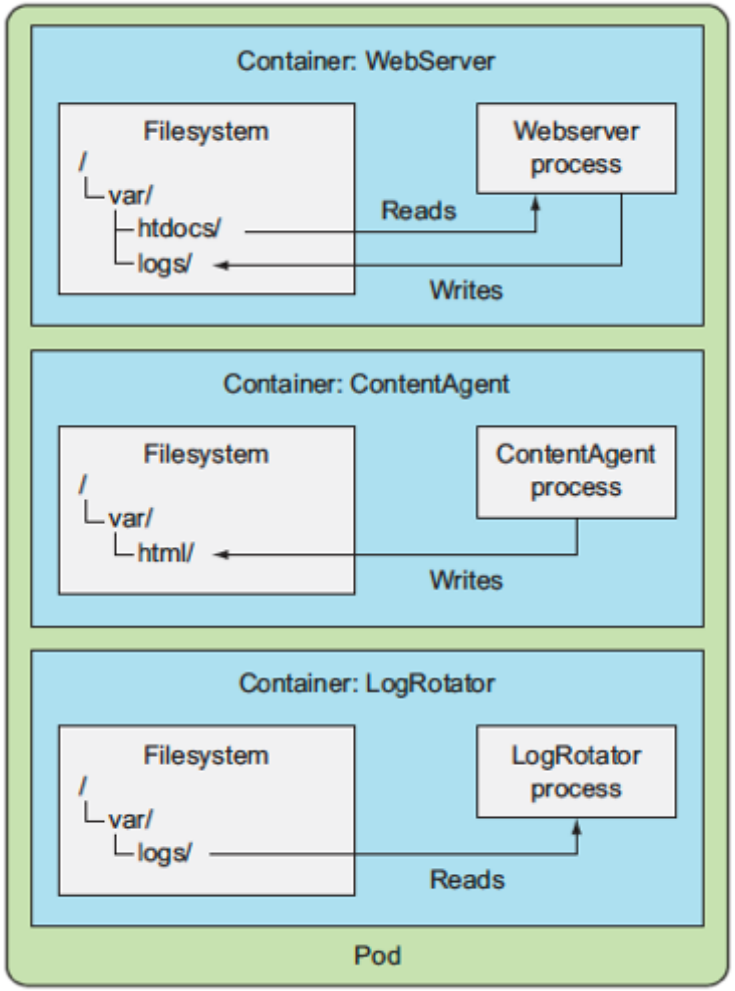
Ingress Load Balancer 개념도(using nginx container)
- domain name based multi-site hosting -

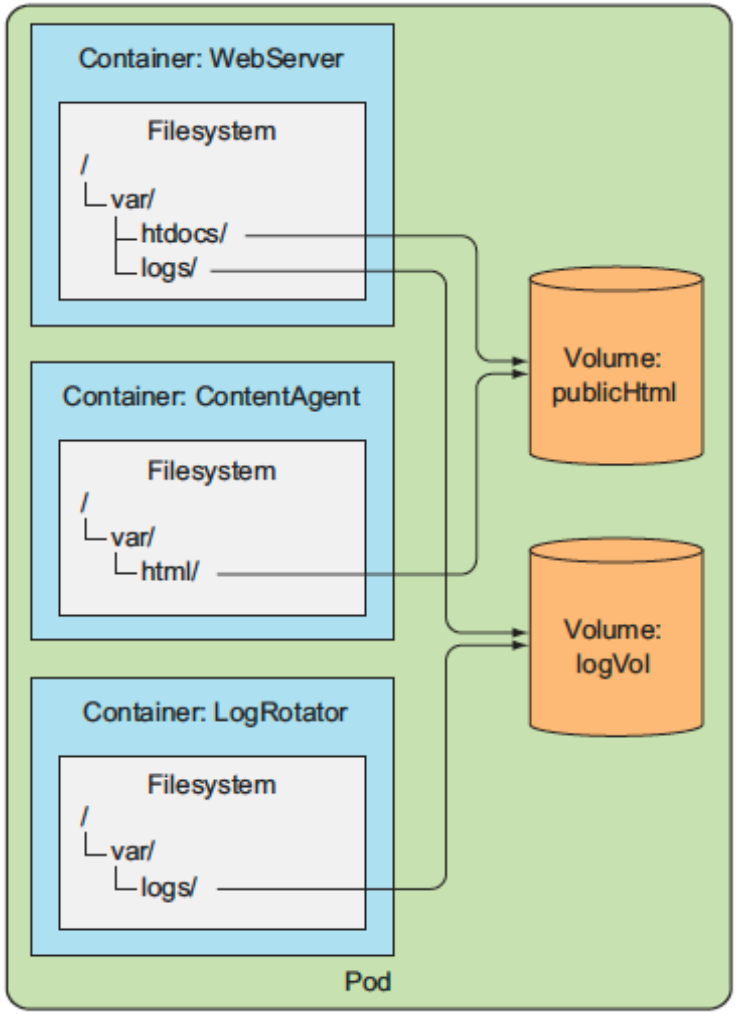


Port Forward

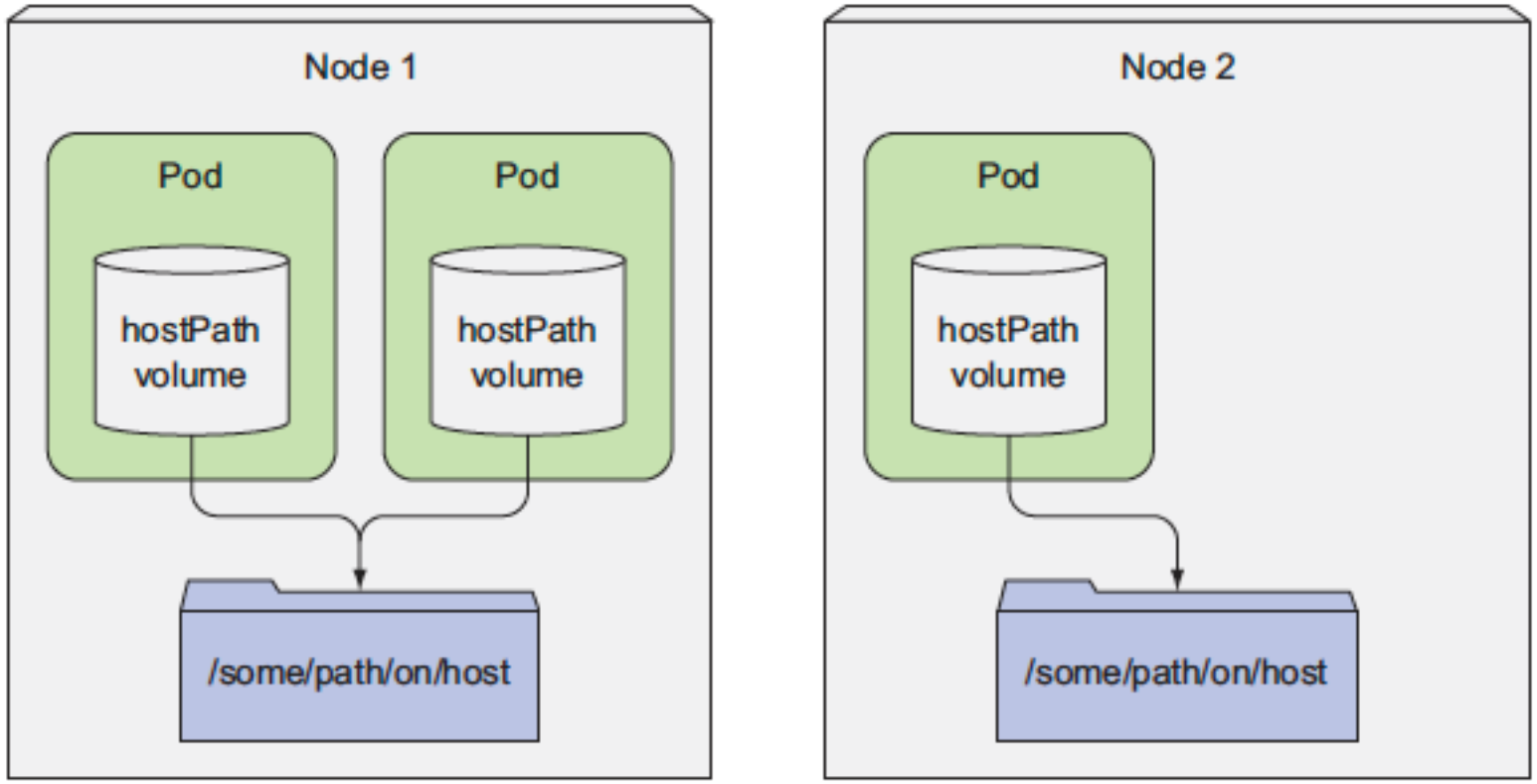


Volume

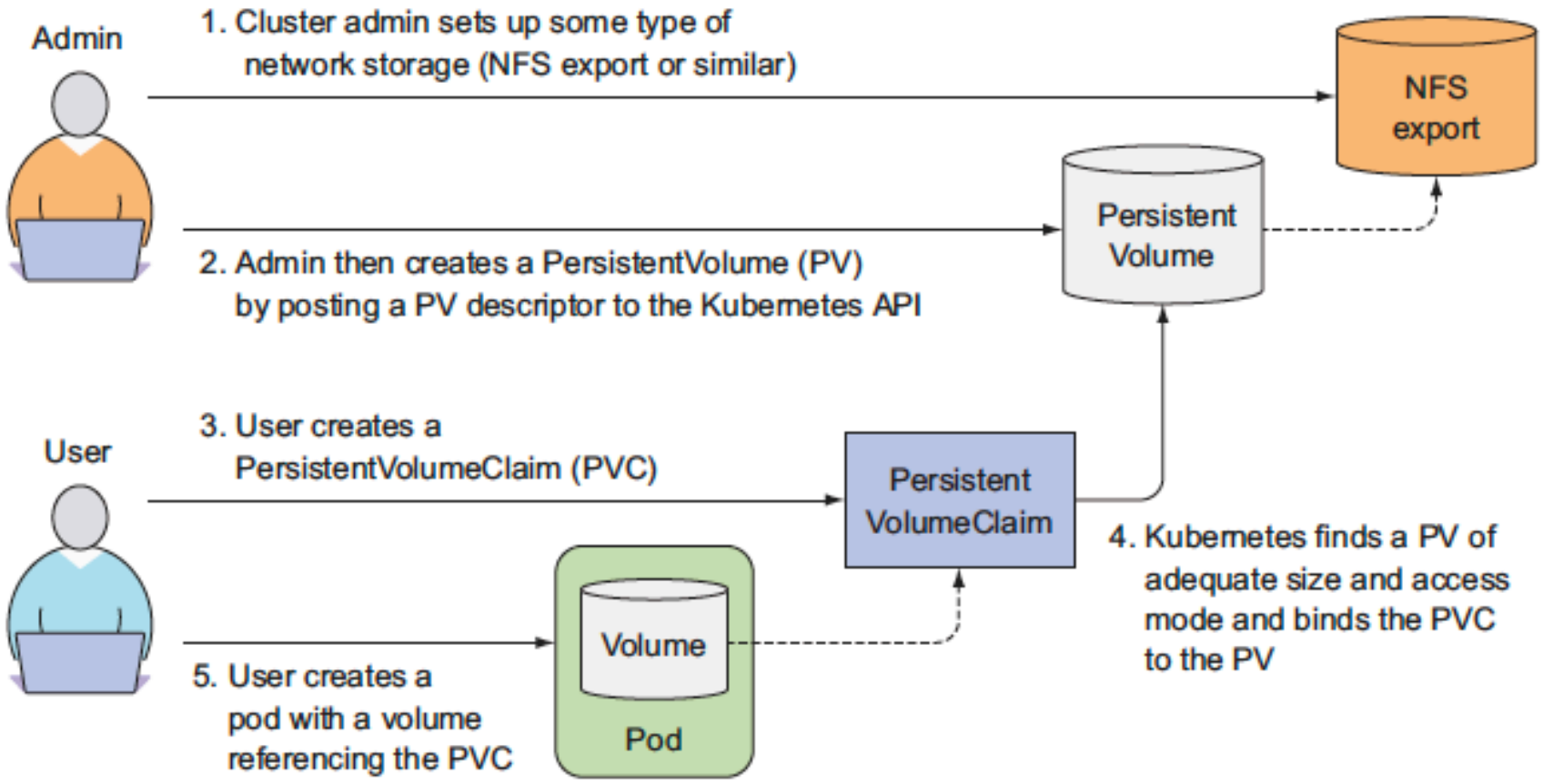

Container: WebServer
Filesystem
/
var/
htdocs/
logs/
Webserver process
Reads
Writes
Container: ContentAgent
Filesystem
/
var/
html/
ContentAgent process
Writes
Container: LogRotator
Filesystem
/
var/
logs/
LogRotator process
Reads
Pod


Container: WebServer
Filesystem
/
var/
htdocs/
logs/
Container: ContentAgent
Filesystem
/
var/
html/
Container: LogRotator
Filesystem
/
var/
logs/
Volume: publicHtml
Volume: logVol
Pod

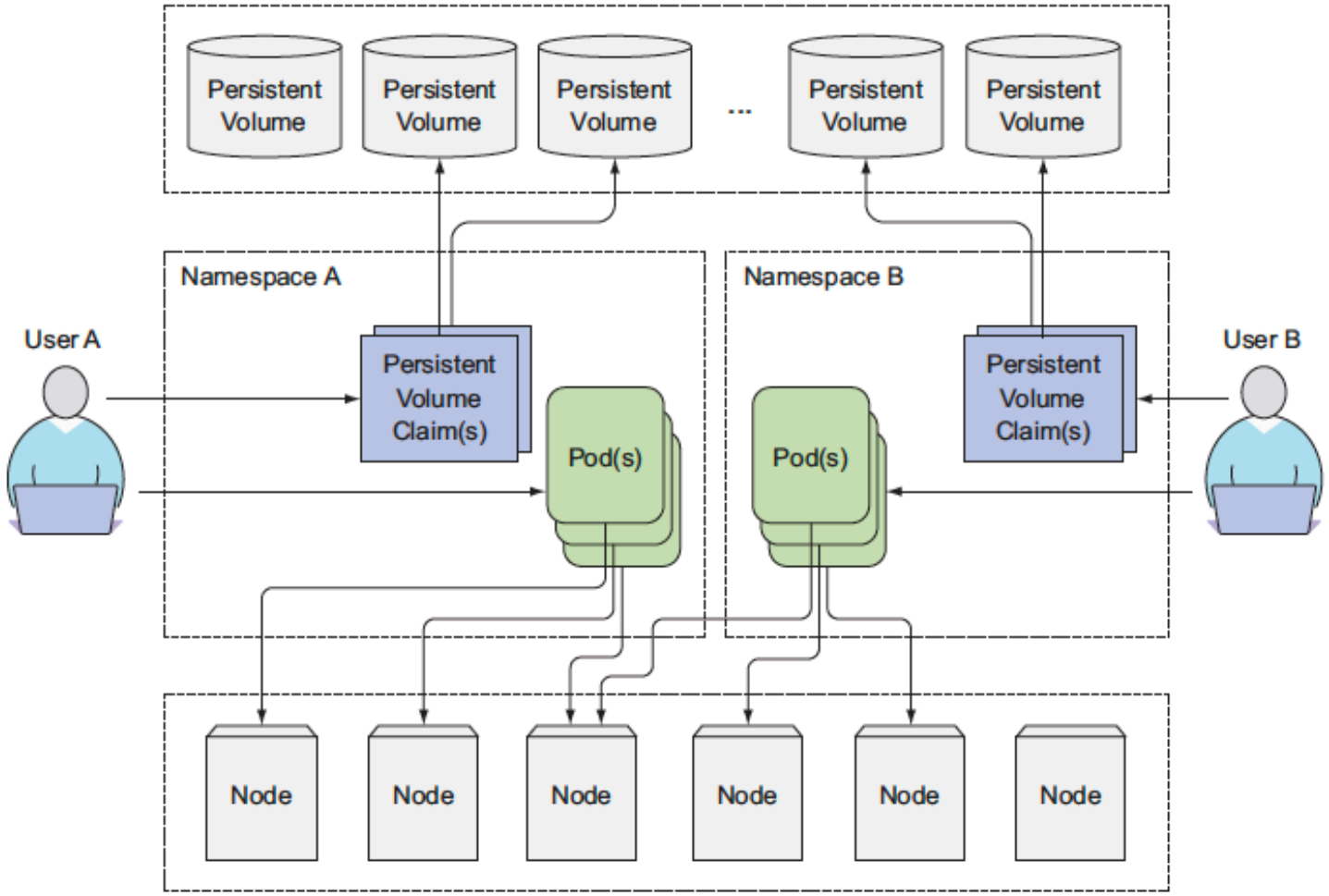
hostPath


Node 1
Pod
hostPath volume
Pod
hostPath volume
/some/path/on/host
Node 2
Pod
hostPath volume
/some/path/on/host

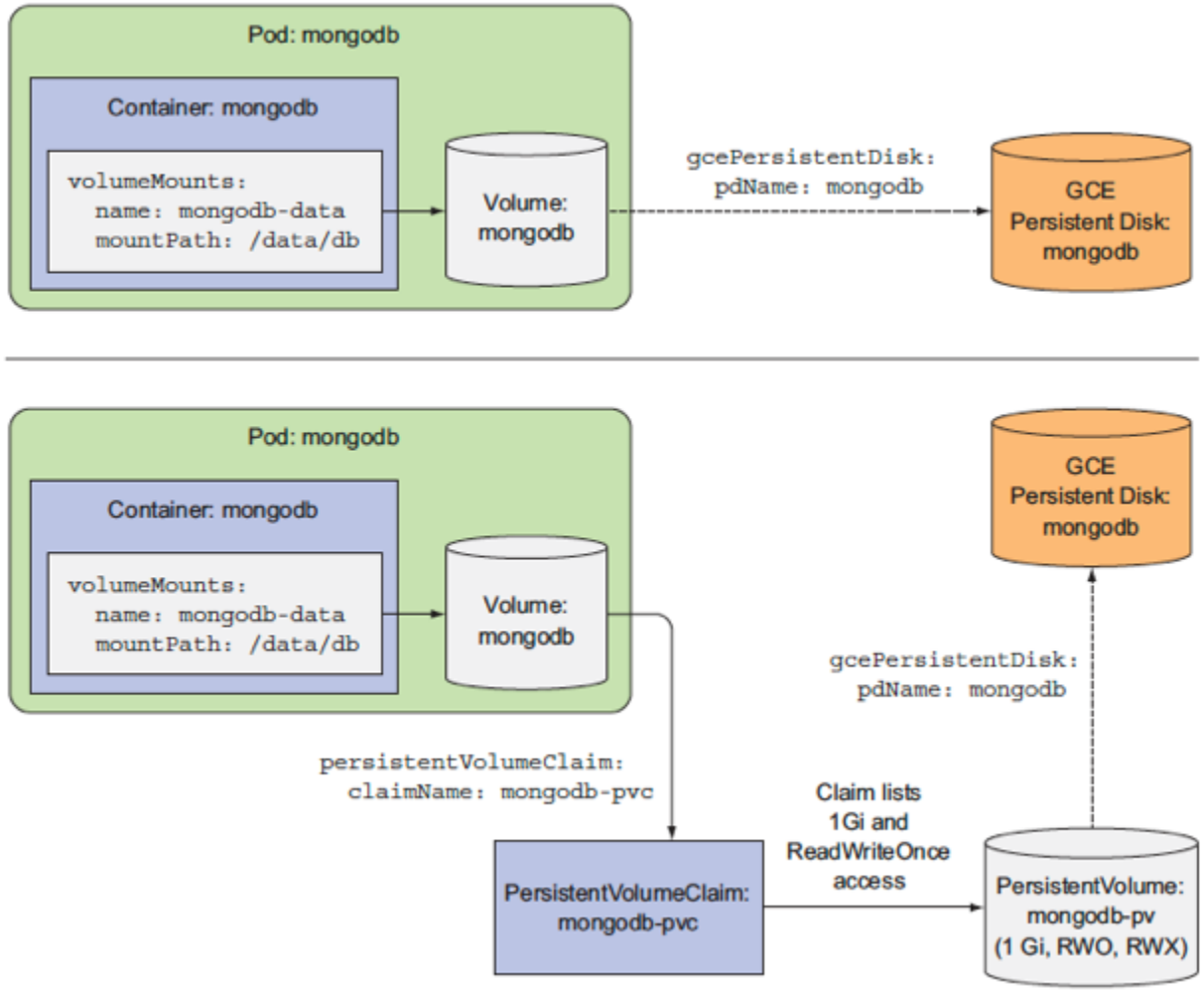
PersistentVolume & PersistentVolumeClaim



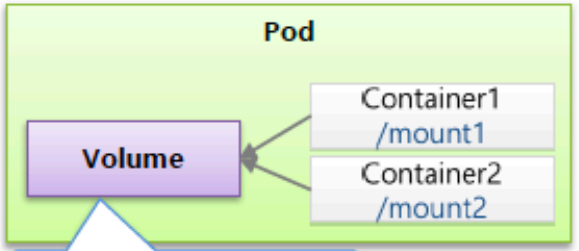
PersistentVolume, Node, Namespace



Use Directly vs Through PV and PVC

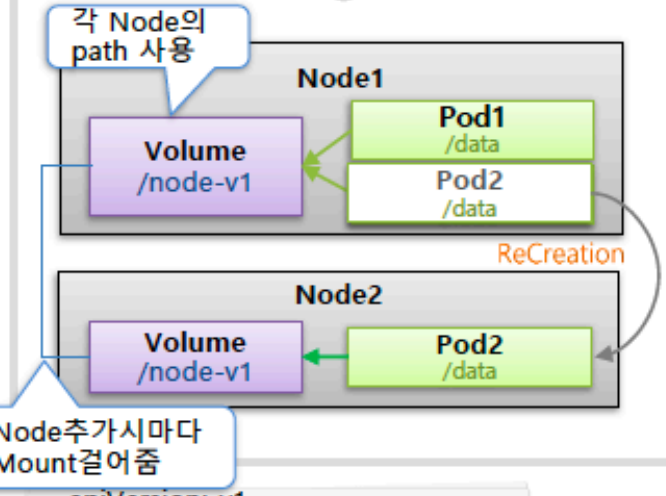


emptyDir




```yaml
apiVersion: v1
kind: Pod
metadata:
 name: pod-volume-1
spec:
 containers:
 - name: container1
   image: tmkube/init
   volumeMounts:
   - name: empty-dir
     mountPath: /mount1
 - name: container2
   image: tmkube/init
   volumeMounts:
   - name: empty-dir
     mountPath: /mount2
 volumes:
 - name : empty-dir
   emptyDir: {}
```

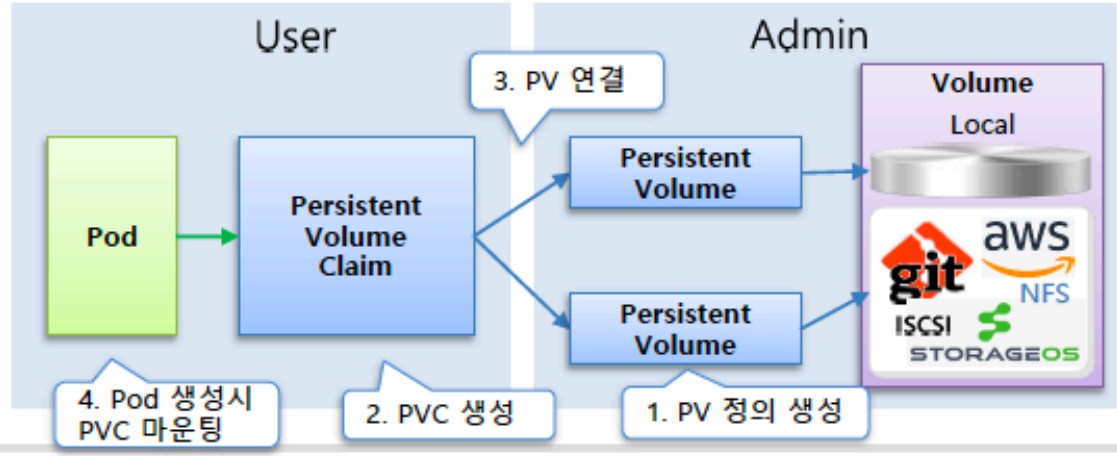
hostPath




```yaml
apiVersion: v1
kind: Pod
metadata:
 name: pod-volume-2
spec:
 containers:
 - name: container
   image: tmkube/init
   volumeMounts:
   - name: host-path
     mountPath: /mount1
 volumes:
 - name : host-path
   hostPath:
     path: /node-v
     type: Directory
```

사전에 해당 Node에
경로가 있어야함

PVC / PV



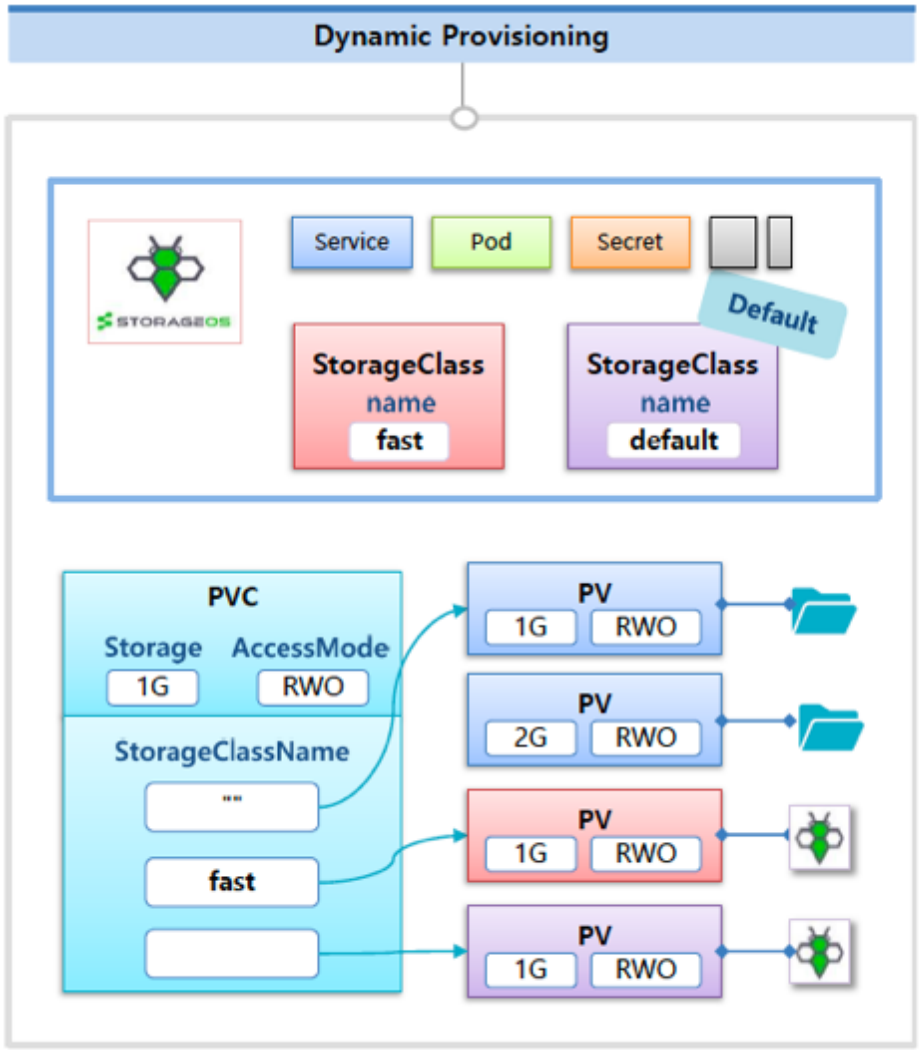

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: pod-volume-3
spec:
 containers:
 - name: container
   image: tmkube/init
   volumeMounts:
   - name: pvc-pv
     mountPath: /volume
 volumes:
 - name : pvc-pv
   persistentVolumeClaim:
     claimName: pvc-01
```

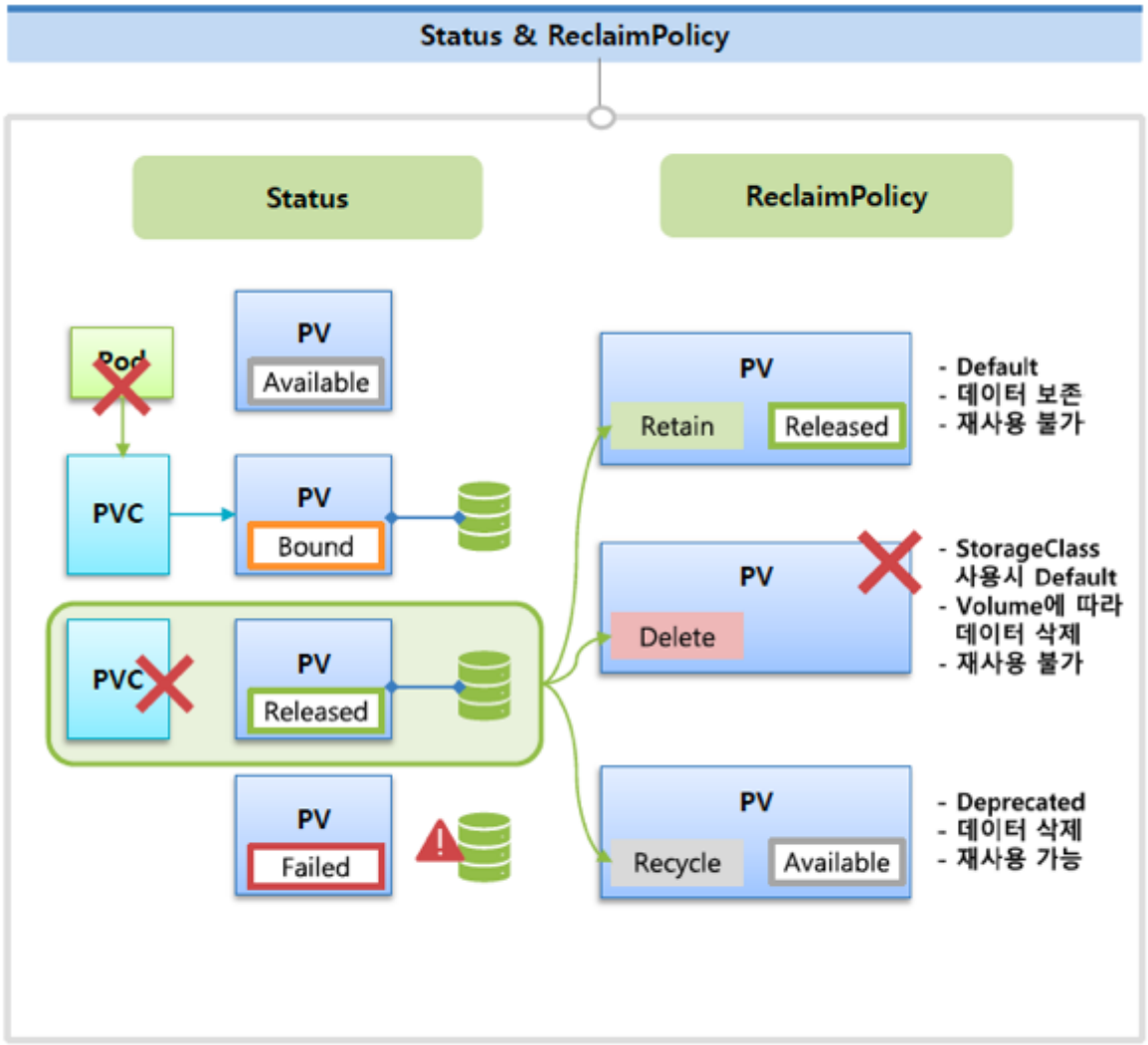
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvc-01
spec:
 accessModes:
   - ReadWriteOnce
 resources:
   requests:
     storage: 1G
 storageClassName: ""
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-01
spec:
  capacity:
    storage: 1G
  accessModes:
    - ReadWriteOnce
  local:
    path: /node-v
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - {key: node, operator:
In, values: [node1]}
```

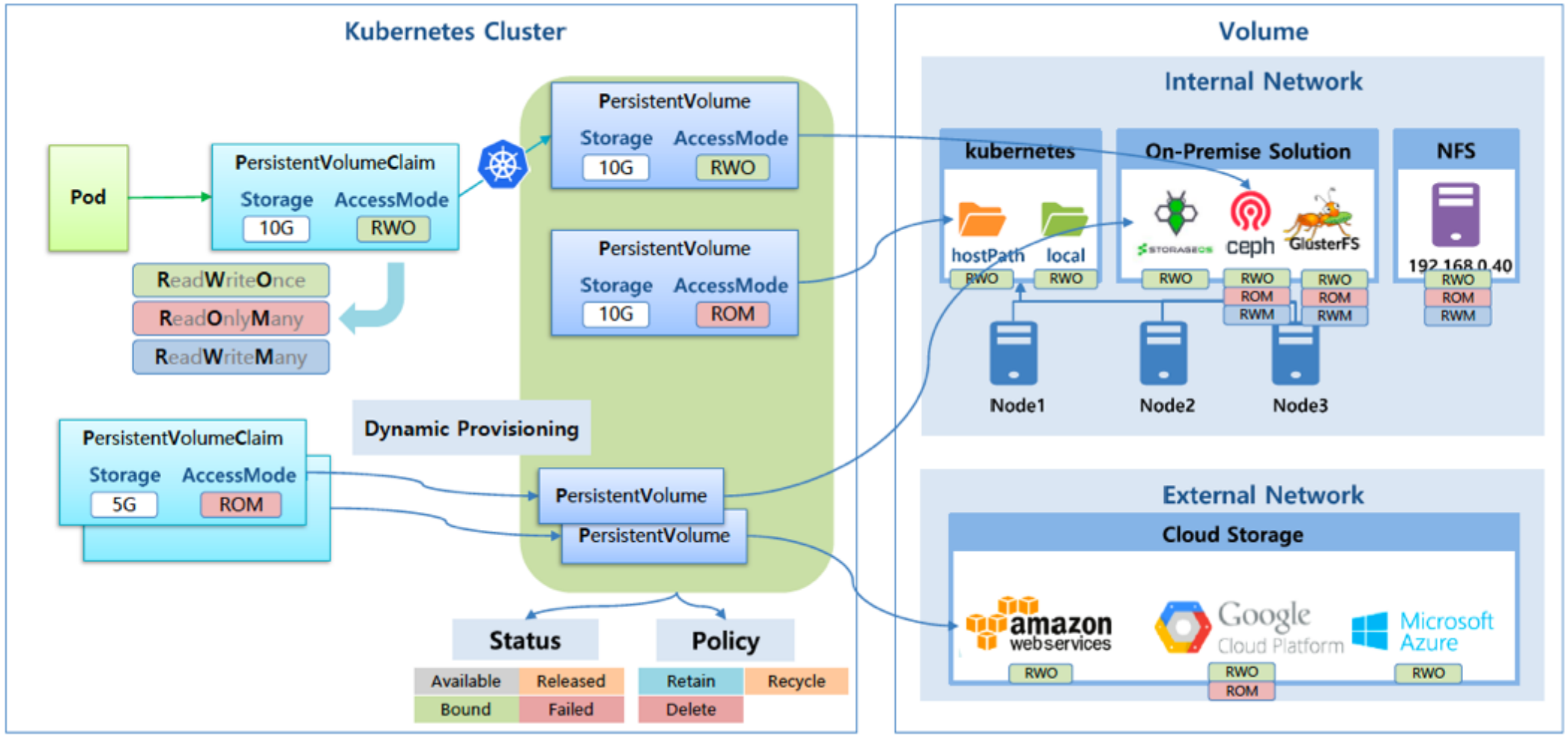
Pod가 해당
Node에 만들어짐

Dynamic Provisioning

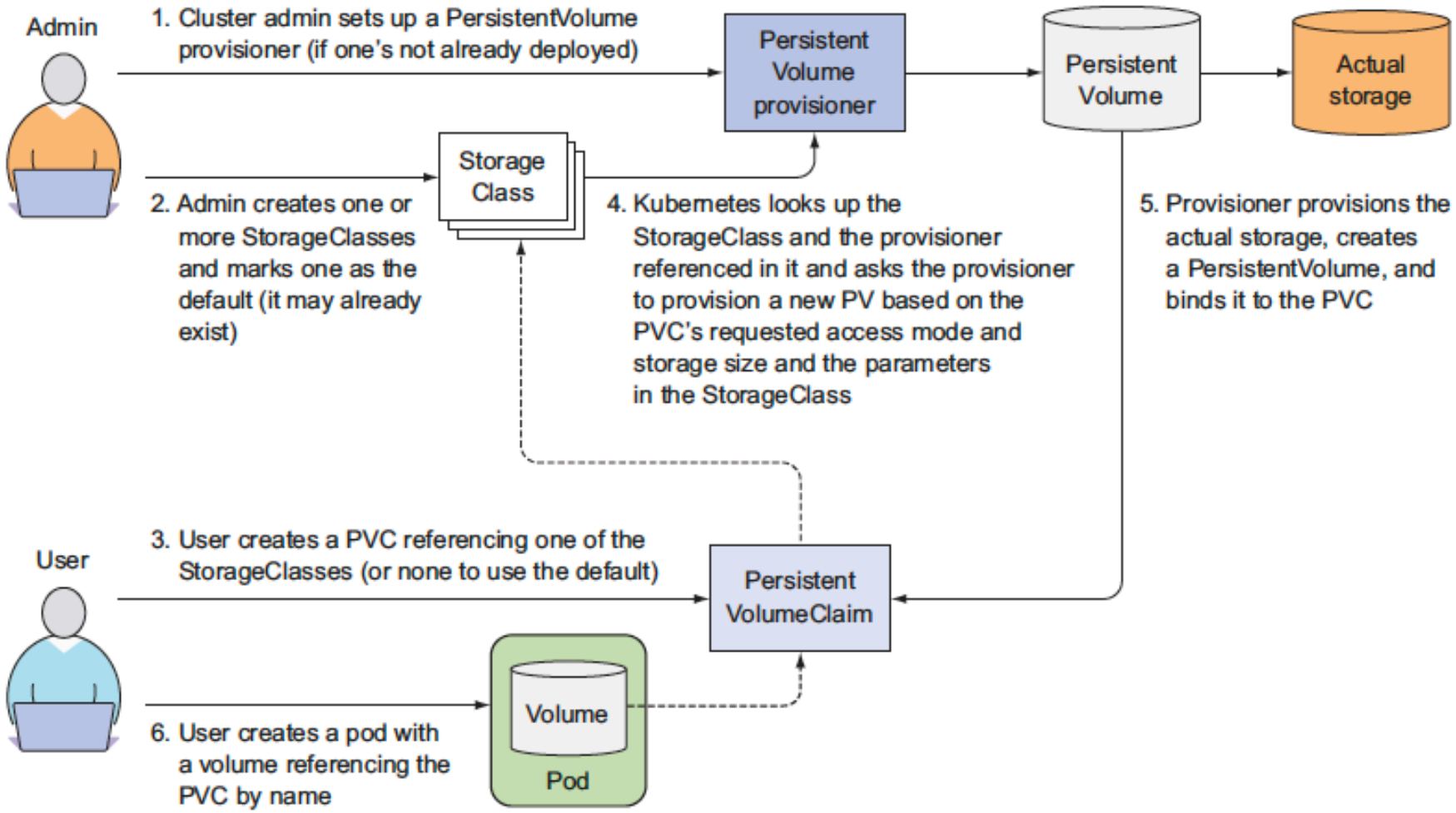


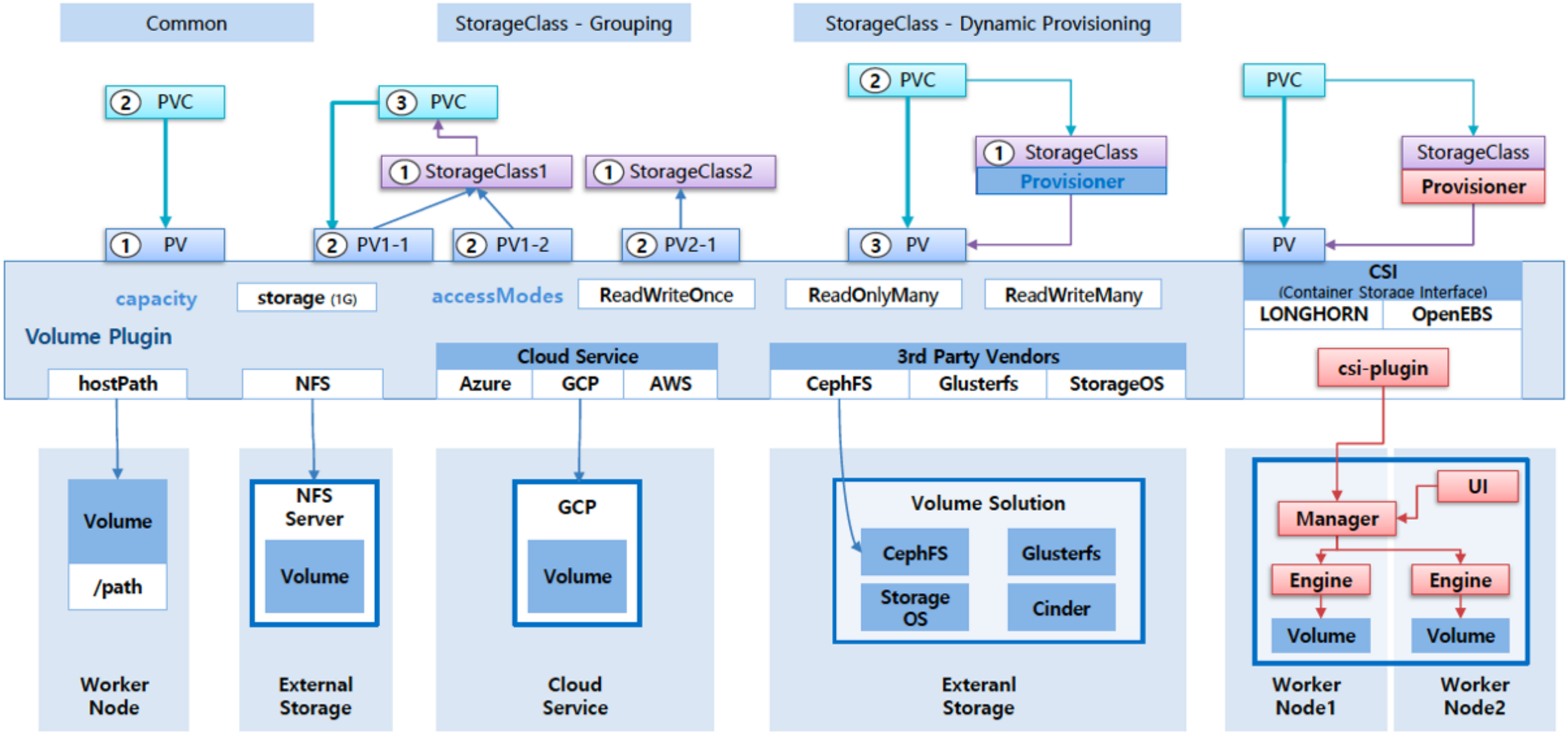


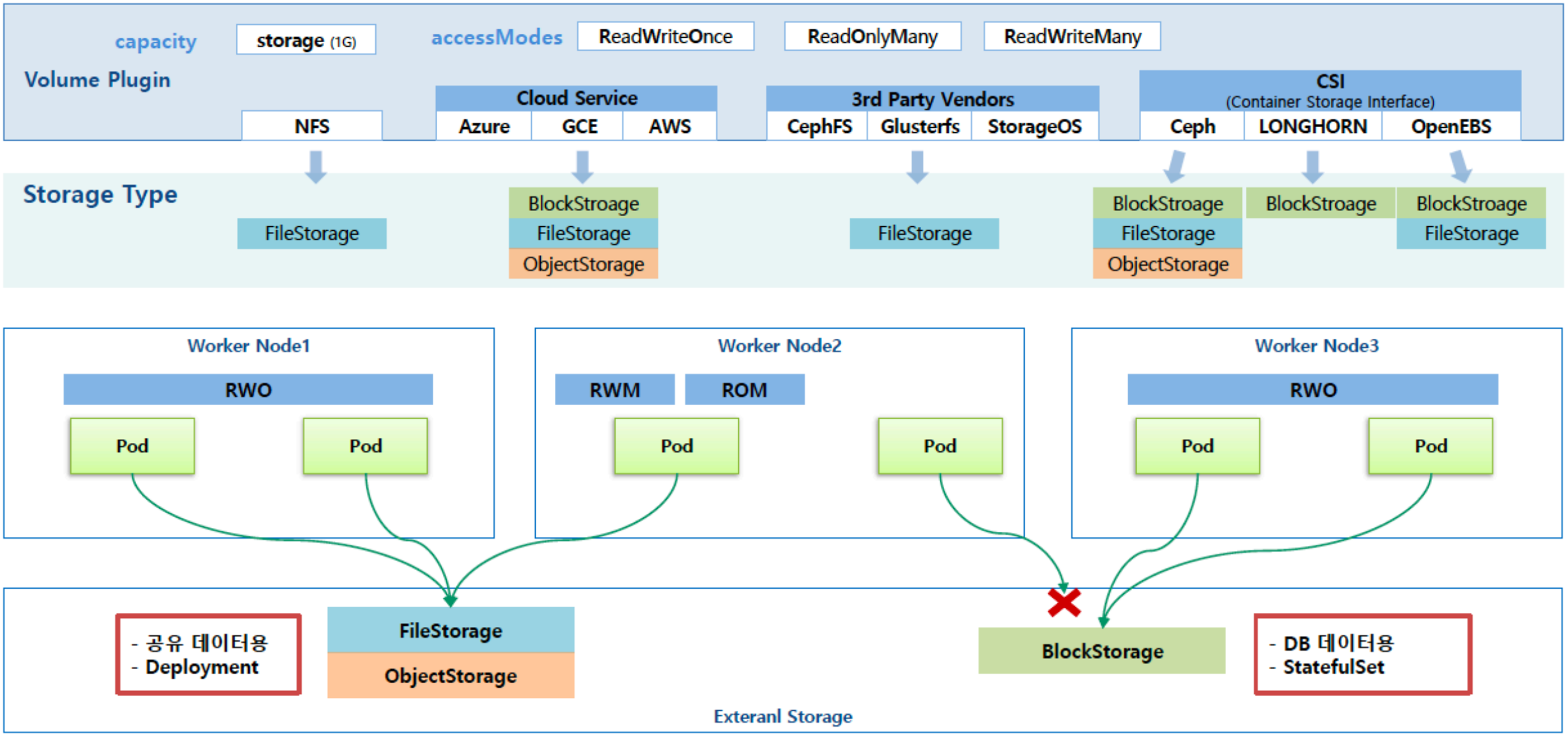
PV, PVC: Complete Picture

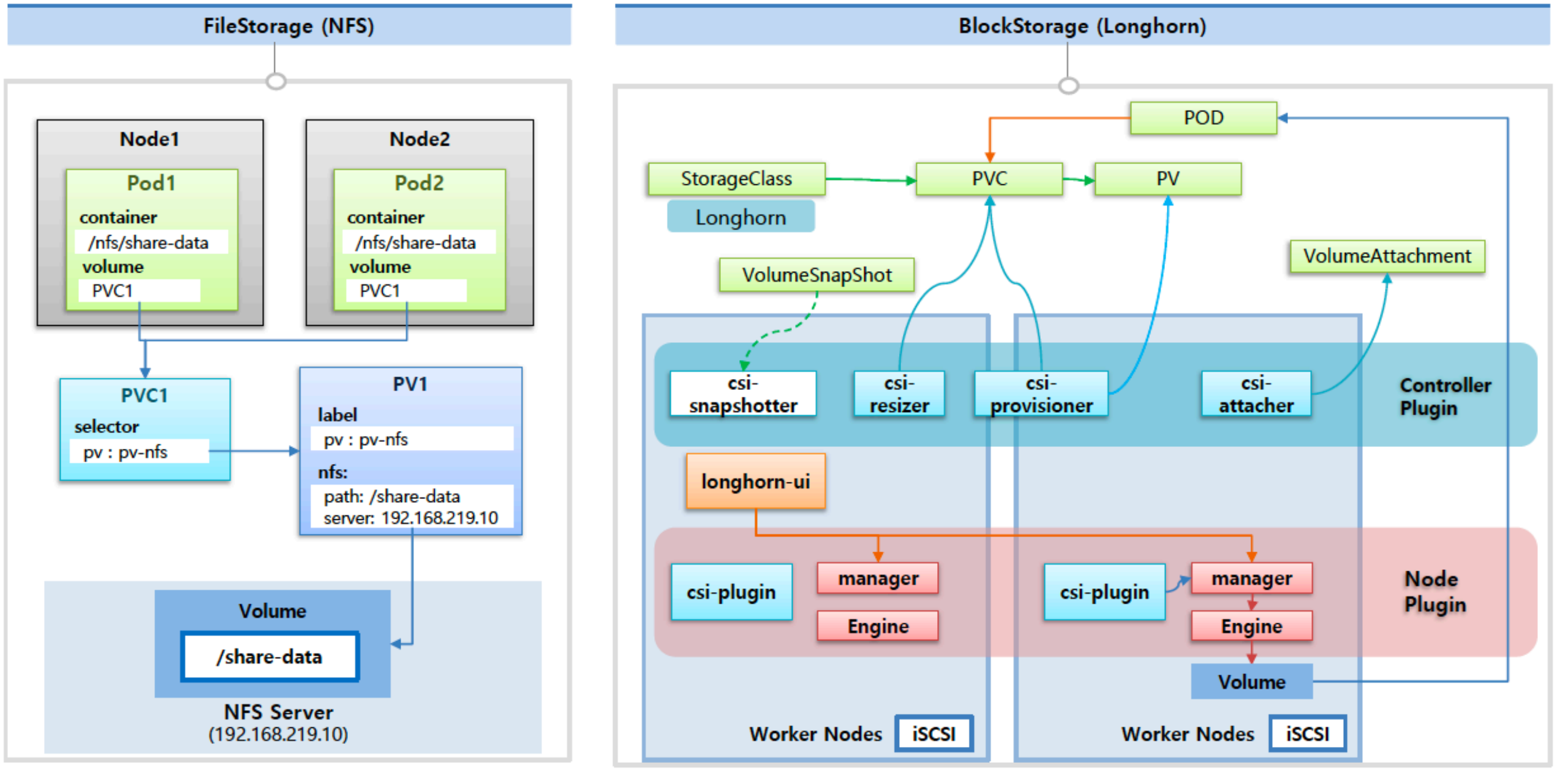


PV, PVC: Complete Steps

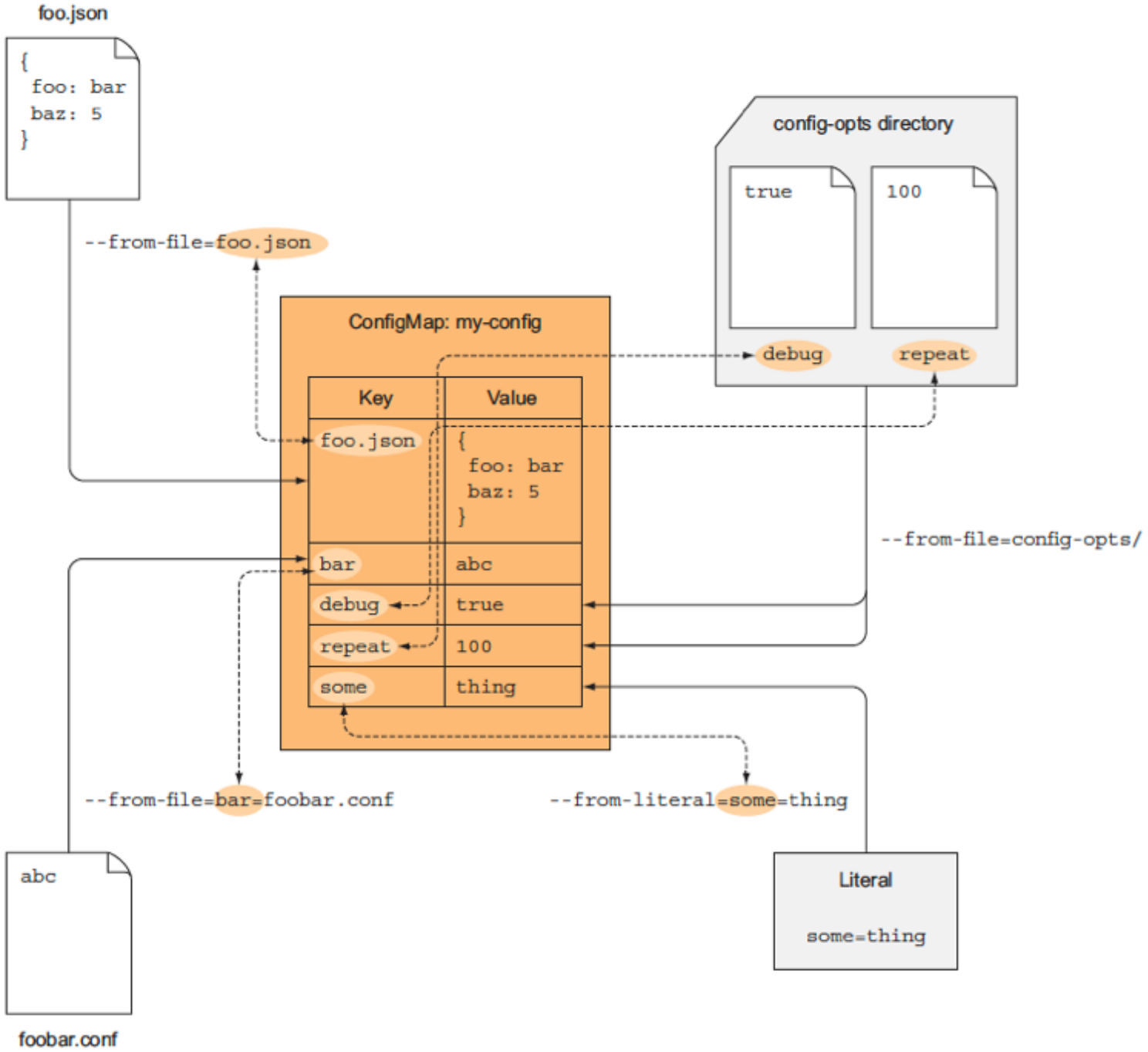



Common
StorageClass - Grouping
StorageClass - Dynamic Provisioning
2 PVC
1 PV
3 PVC
1 StorageClass1
1 StorageClass2
2 PV1-1
2 PV1-2
2 PV2-1
2 PVC
1 StorageClass
Provisioner
3 PV
PVC
StorageClass
Provisioner
PV
capacity
storage (1G)
accessModes
ReadWriteOnce
ReadOnlyMany
ReadWriteMany
Volume Plugin
CSI
(Container Storage Interface)
LONGHORN
OpenEBS
csi-plugin
hostPath
NFS
Cloud Service
Azure
GCP
AWS
3rd Party Vendors
CephFS
Glusterfs
StorageOS
Volume
/path
NFS Server
Volume
GCP
Volume
Volume Solution
CephFS
Glusterfs
Storage OS
Cinder
UI
Manager
Engine
Engine
Volume
Volume
Worker Node
External Storage
Cloud Service
Exteranl Storage
Worker Node1
Worker Node2

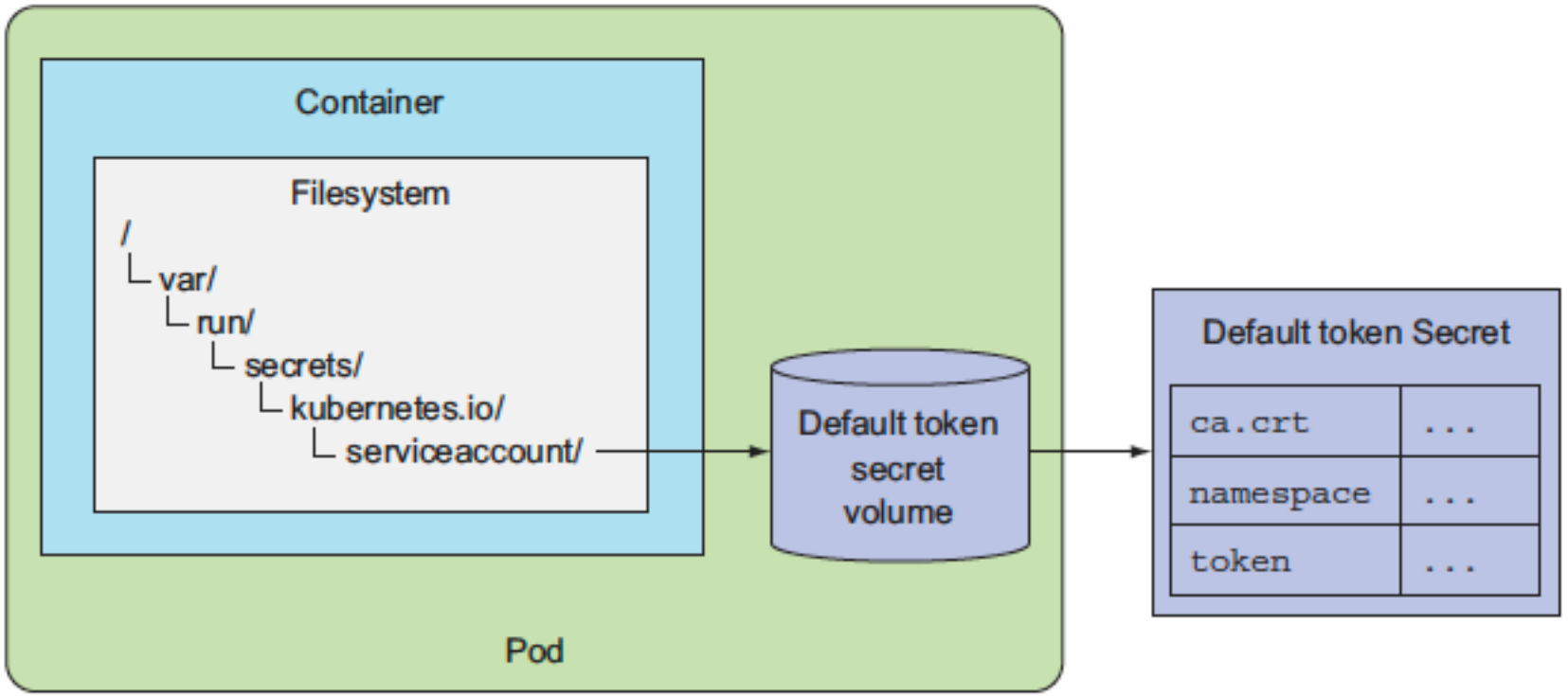

capacity
storage (1G)
accessModes
ReadWriteOnce
ReadOnlyMany
ReadWriteMany
Volume Plugin
NFS
Cloud Service
Azure
GCE
AWS
3rd Party Vendors
CephFS
Glusterfs
StorageOS
CSI
(Container Storage Interface)
Ceph
LONGHORN
OpenEBS
Storage Type
FileStorage
BlockStroage
FileStorage
ObjectStorage
FileStorage
BlockStroage
FileStorage
ObjectStorage
BlockStroage
BlockStroage
FileStorage
Worker Node1
RWO
Pod
Pod
Worker Node2
RWM
ROM
Pod
Pod
Worker Node3
RWO
Pod
Pod
- 공유 데이터용
- Deployment
FileStorage
ObjectStorage
BlockStorage
- DB 데이터용
- StatefulSet
Exteranl Storage


FileStorage (NFS)
Node1
Pod1
container
/nfs/share-data
volume
PVC1
Node2
Pod2
container
/nfs/share-data
volume
PVC1
PVC1
selector
pv : pv-nfs
PV1
label
pv : pv-nfs
nfs:
path: /share-data
server: 192.168.219.10
Volume
/share-data
NFS Server
(192.168.219.10)
BlockStorage (Longhorn)
POD
StorageClass
Longhorn
PVC
PV
VolumeSnapShot
VolumeAttachment
csi-snapshotter
csi-resizer
csi-provisioner
csi-attacher
Controller Plugin
longhorn-ui
csi-plugin
manager
Engine
csi-plugin
manager
Engine
Node Plugin
Volume
Worker Nodes
iSCSI
Worker Nodes
iSCSI

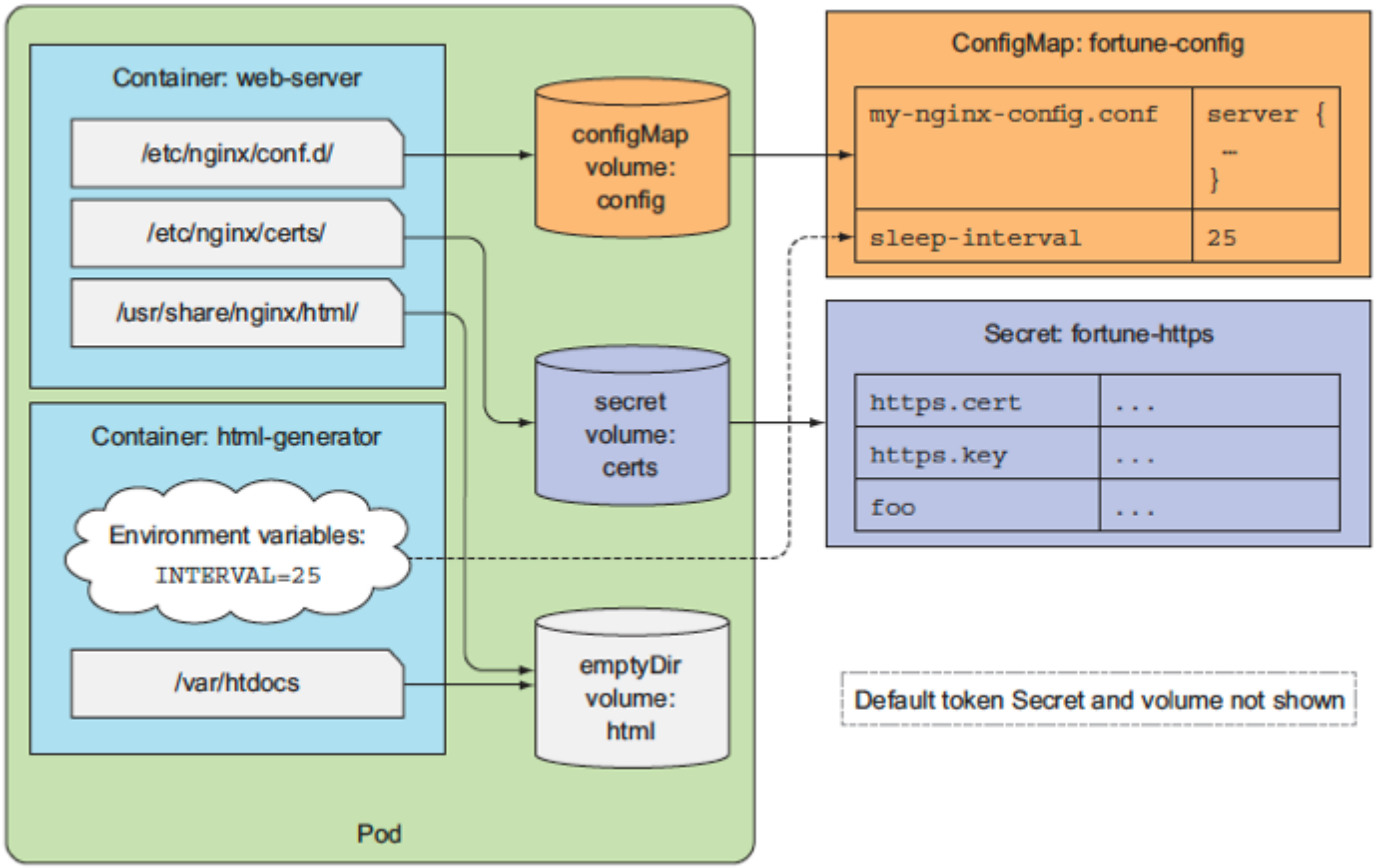
ConfigMap


foo.json
{
foo: bar
baz: 5
}
--from-file=foo.json
config-opts directory
true
100
debug
repeat
ConfigMap: my-config
Key
Value
foo.json
{
foo: bar
baz: 5
}
bar
abc
debug
true
repeat
100
some
thing
--from-file=config-opts/
--from-file=bar=foobar.conf
--from-literal=some=thing
abc
foobar.conf
Literal
some=thing

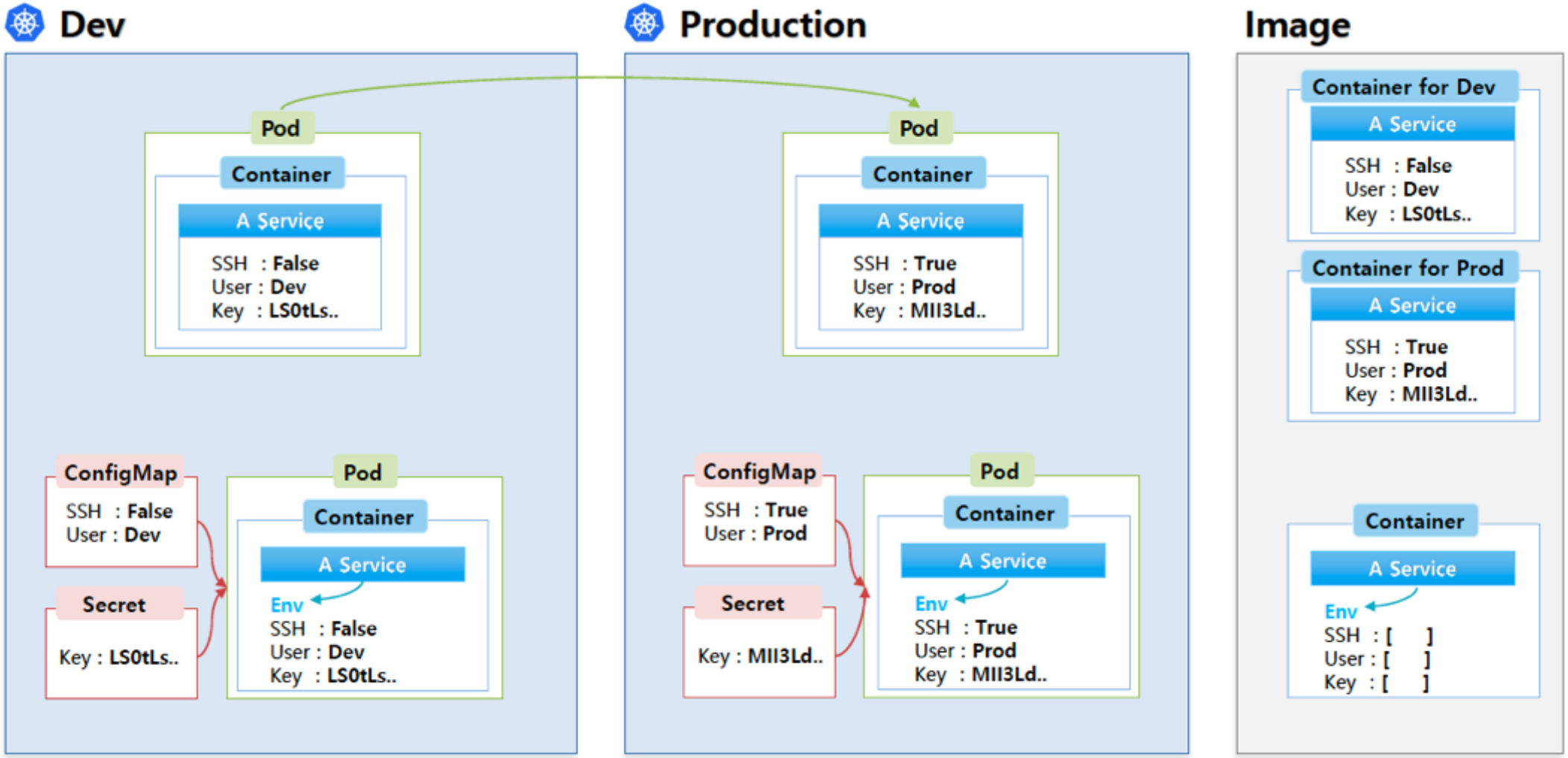
Secret (Default Token)



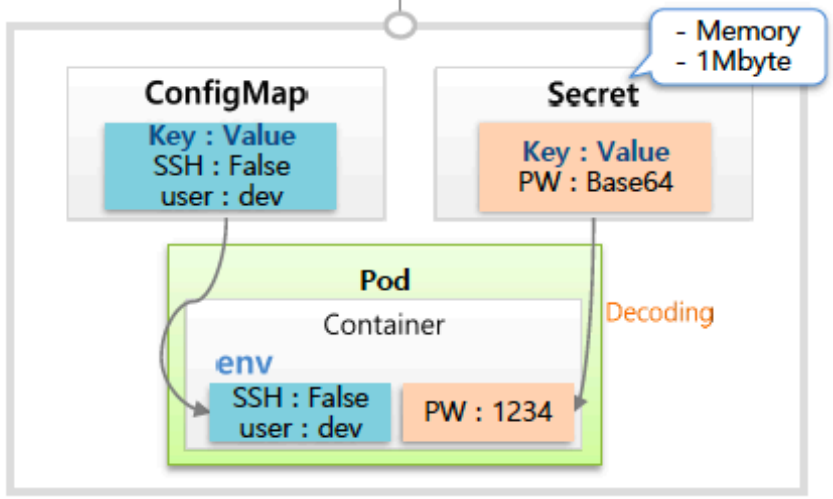
ConfigMap & Secret



ConfigMap & Secret



Env (Literal)

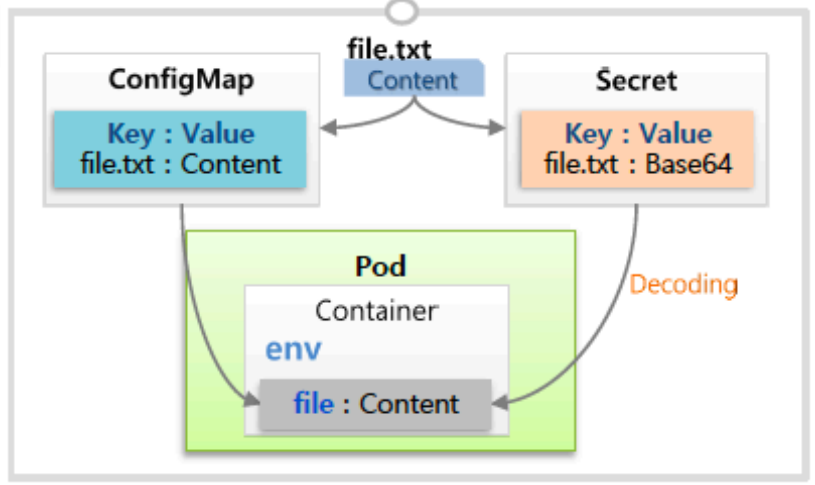



```
apiVersion: v1
kind: ConfigMap
metadata:
 name: cm-dev
data:
 SSH: False
 User: dev
```

```
apiVersion: v1
kind: Secret
metadata:
 name: sec-dev
data:
 Key: MTIzNA==
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
spec:
  containers:
    - name: container
      image: tmkube/init
      envFrom:
      - configMapRef:
          name: cm-dev
      - secretRef:
          name: sec-dev
```

Env (File)

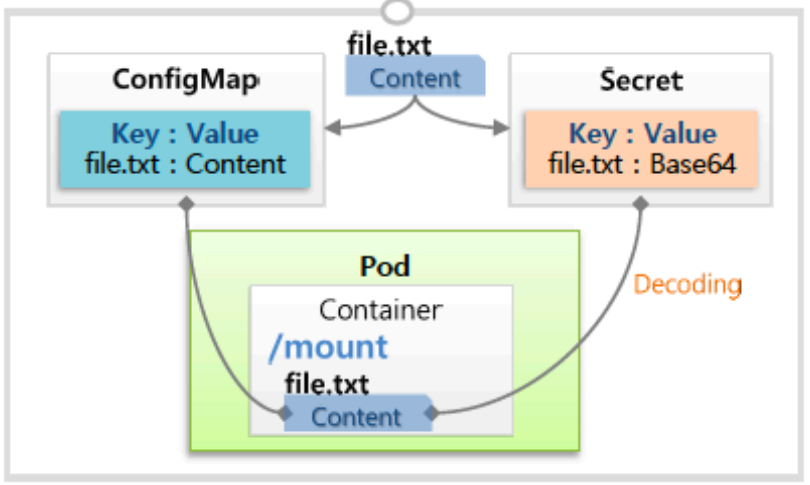



```
kubectl create configmap cm-file
--from-file=./file.txt
```

```
kubectl create secret generic sec-file
--from-file=./file.txt
```

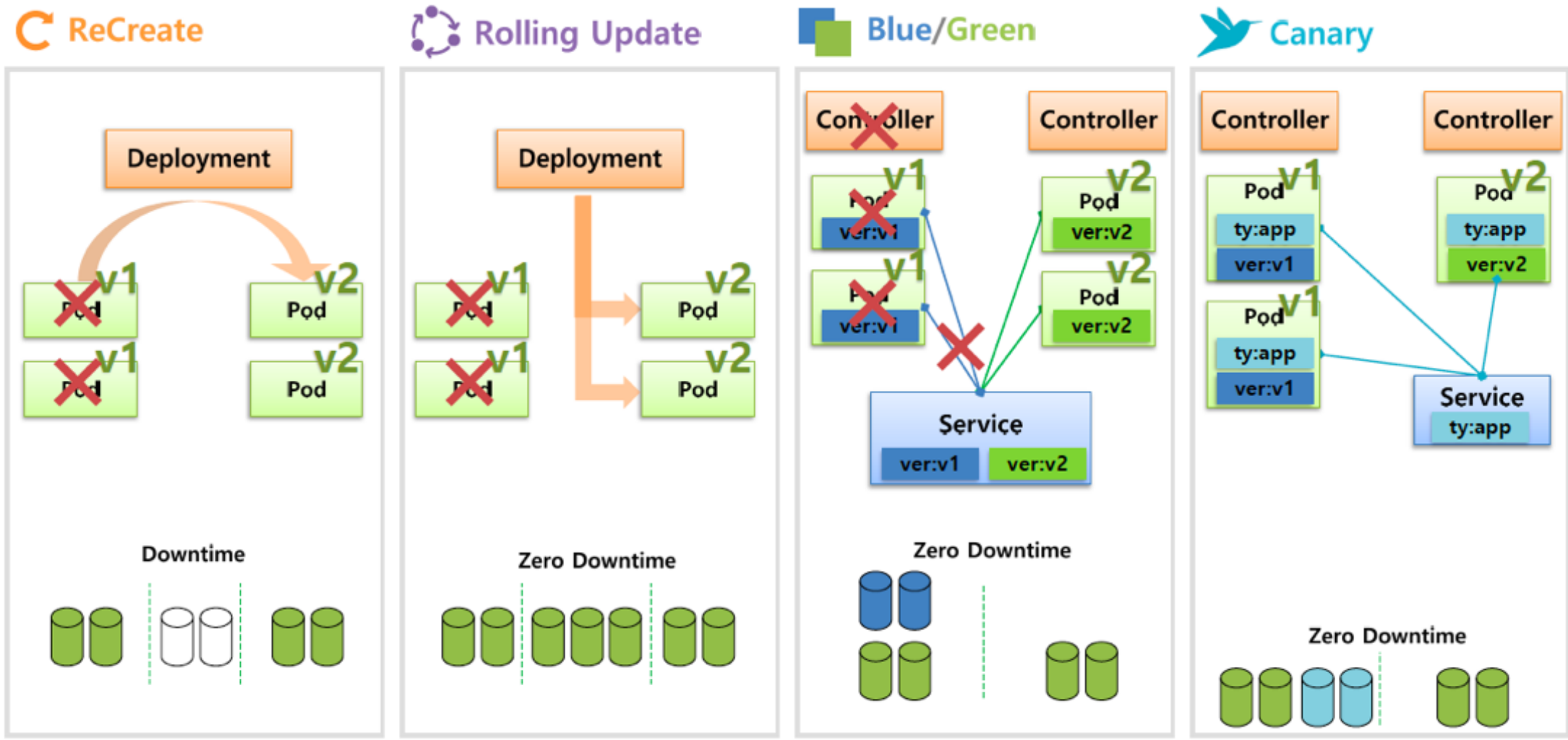
```
apiVersion: v1
kind: Pod
metadata:
  name: file
spec:
  containers:
    - name: container
      image: tmkube/init
      env:
        - name: file
          valueFrom:
            configMapKeyRef:
              name: cm-file
              key: file.txt
```

Volume Mount (File)

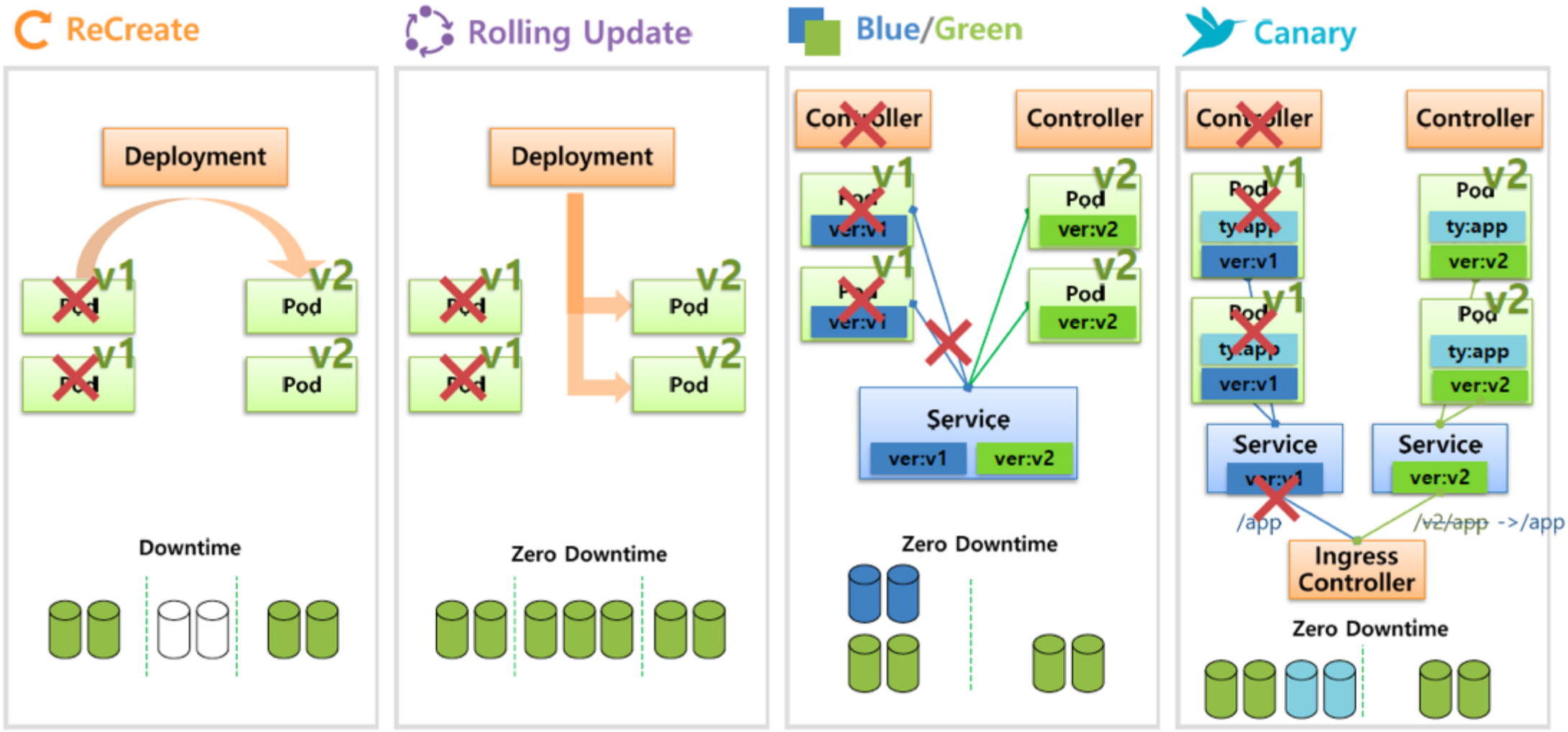



```
apiVersion: v1
kind: Pod
metadata:
  name: mount
spec:
  containers:
    - name: container
      image: tmkube/init
      volumeMounts:
      - name: file-volume
        mountPath: /mount
  volumes:
  - name: file-volume
    configMap:
      name: cm-file
```

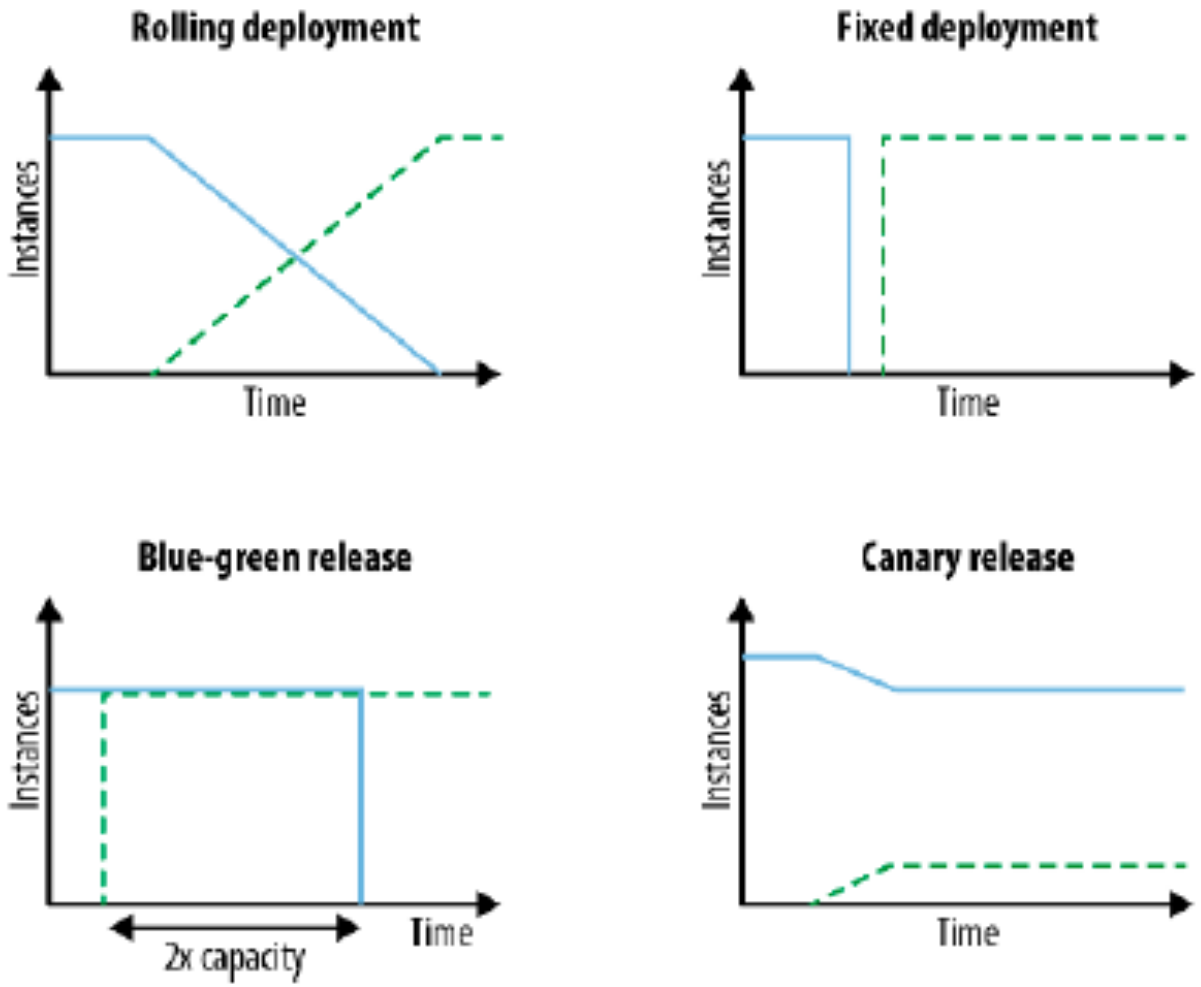
Deployment


ReCreate
Deployment
v1
Pod
v2
Pod
Downtime
Rolling Update
Deployment
v1
Pod
v2
Pod
Zero Downtime
Blue/Green
Controller
Controller
v1
ver:v1
v2
Pod
ver:v2
Service
ver:v1
ver:v2
Zero Downtime
Canary
Controller
Controller
Pod
ty:app
ver:v1
ver:v2
Service
ty:app
Zero Downtime

Deployment


ReCreate
Deployment
v1
Pod
v2
Pod
Downtime
Rolling Update
Deployment
v1
Pod
v2
Pod
Zero Downtime
Blue/Green
Controller
Controller
v1
ver:v1
v2
Pod
ver:v2
Service
ver:v1
ver:v2
Zero Downtime
Canary
Controller
Controller
v1
Pod
ver:v1
v2
Pod
ty:app
ver:v2
Service
Service
ver:v2
/app
/v2/app ->/app
Ingress Controller
Zero Downtime

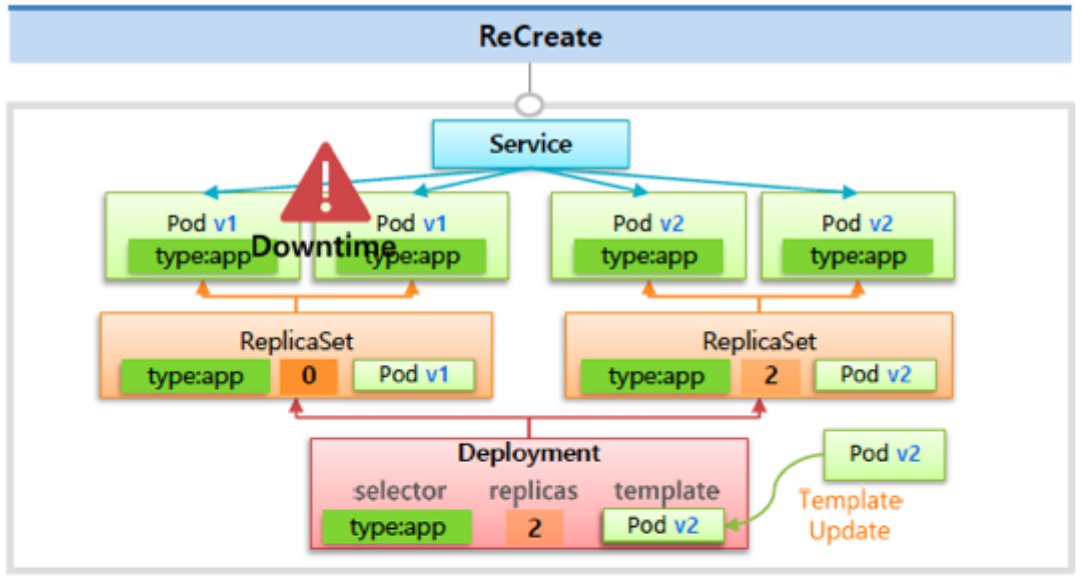
Deployment Strategies

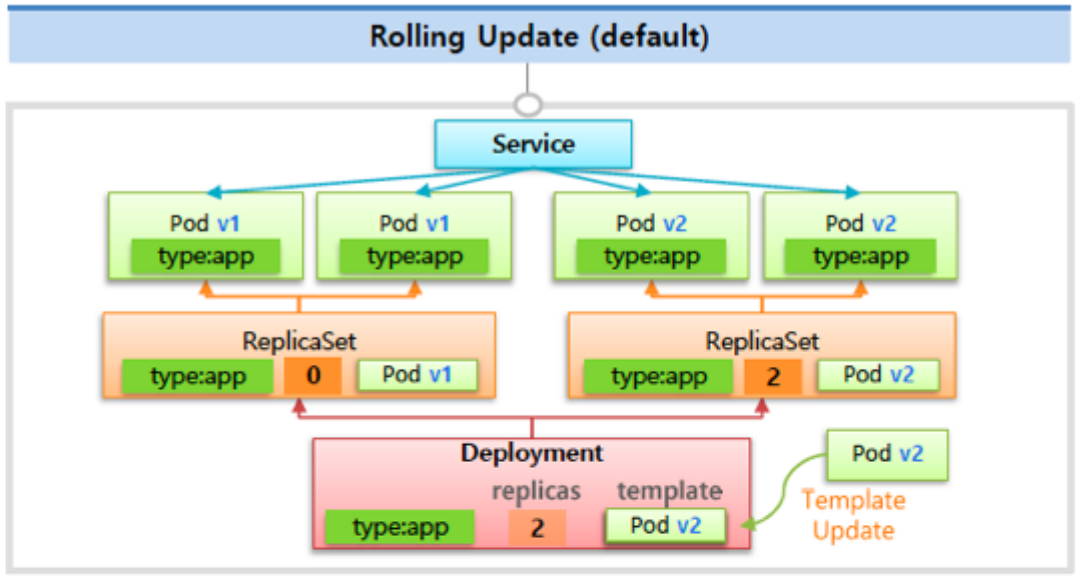


Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

https://github.com/ContainerSolutions/k8s-deployment-strategies

ReCreate, Rolling Update: YAML





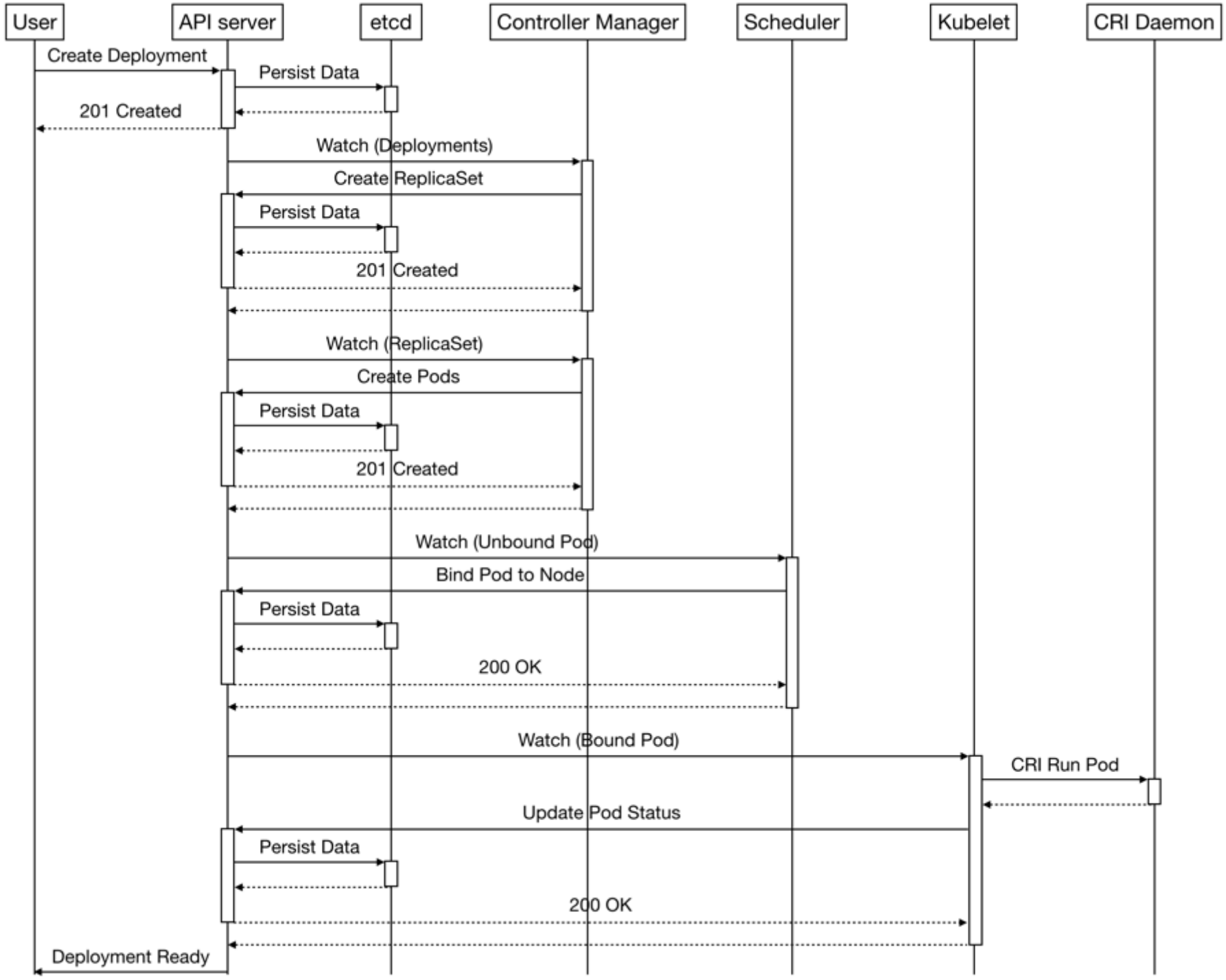

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-1
spec:
  selector:
    type: app
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-1
spec:
  selector:
    matchLabels:
      type: app
  replicas: 2
  strategy:
    type: Recreate
  revisionHistoryLimit: 1
  template:
    metadata:
      labels:
        type: app
    spec:
      containers:
      - name: container
        image: tmkube/app:v1
```

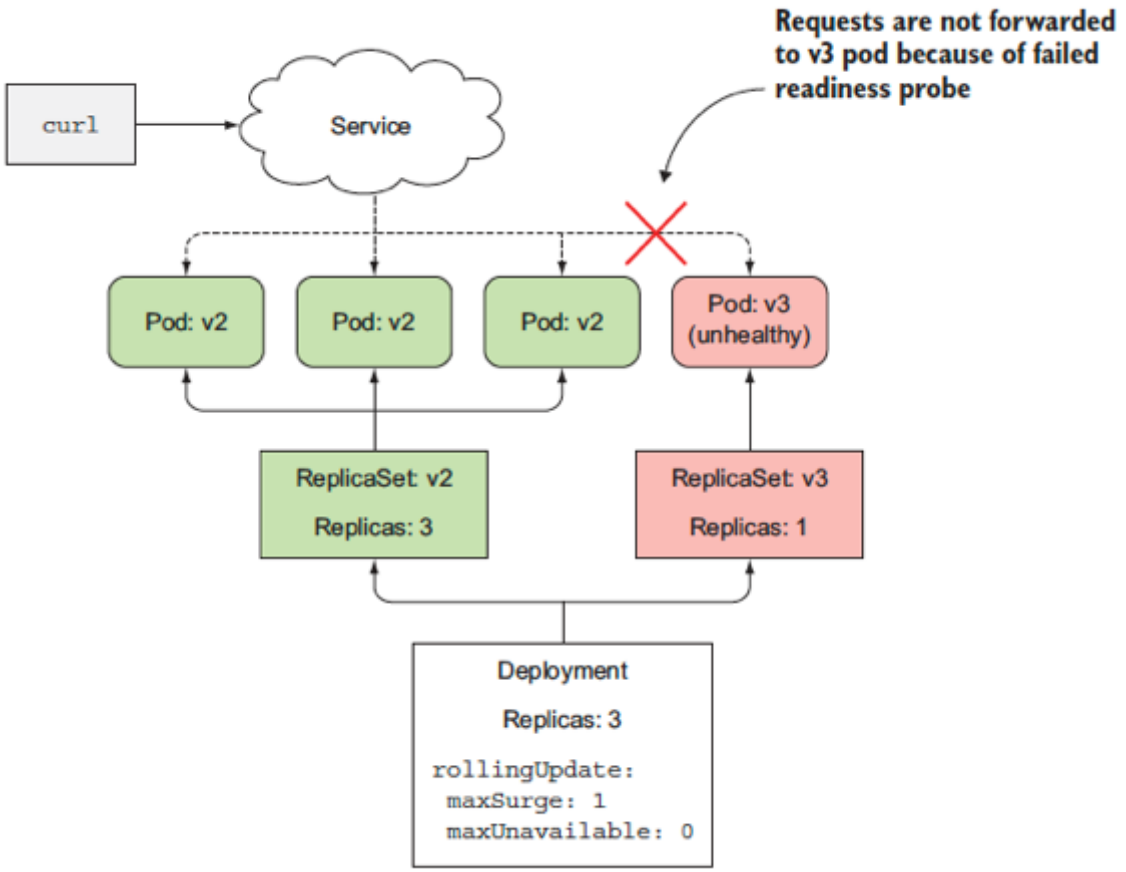
```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-2
spec:
  selector:
    type: app
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-2
spec:
  selector:
    matchLabels:
      type: app
  replicas: 2
  strategy:
    type: RollingUpdate
  minReadySeconds: 10
  template:
    metadata:
      labels:
        type: app
    spec:
      containers:
      - name: container
        image: tmkube/app:v1
```

Deployment Flow Diagram



Deployment + readinessProbe




```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  minReadySeconds: 10
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v3
      name: nodejs
      readinessProbe:
        periodSeconds: 1
        httpGet:
          path: /
          port: 8080
```

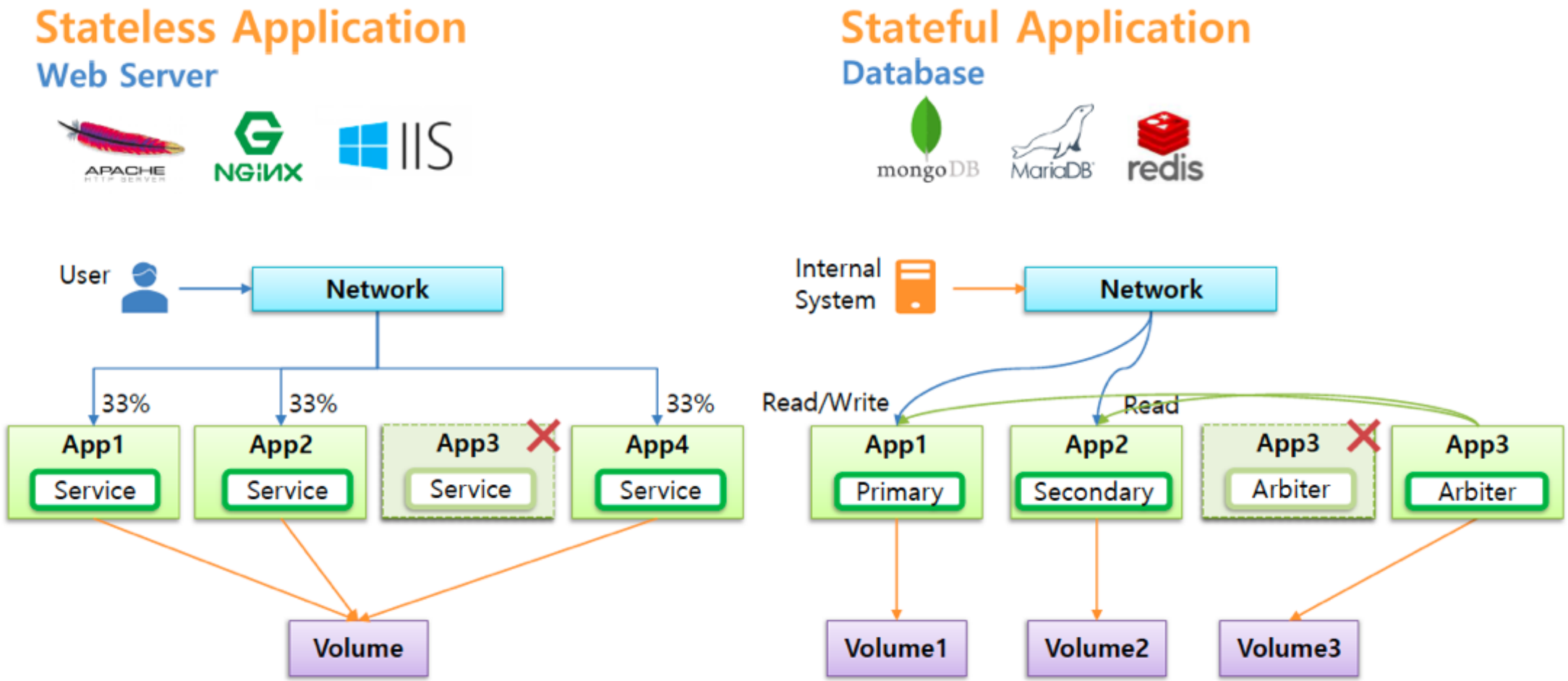
You're keeping minReadySeconds set to 10.

You're keeping maxUnavailable set to 0 to make the deployment replace pods one by one

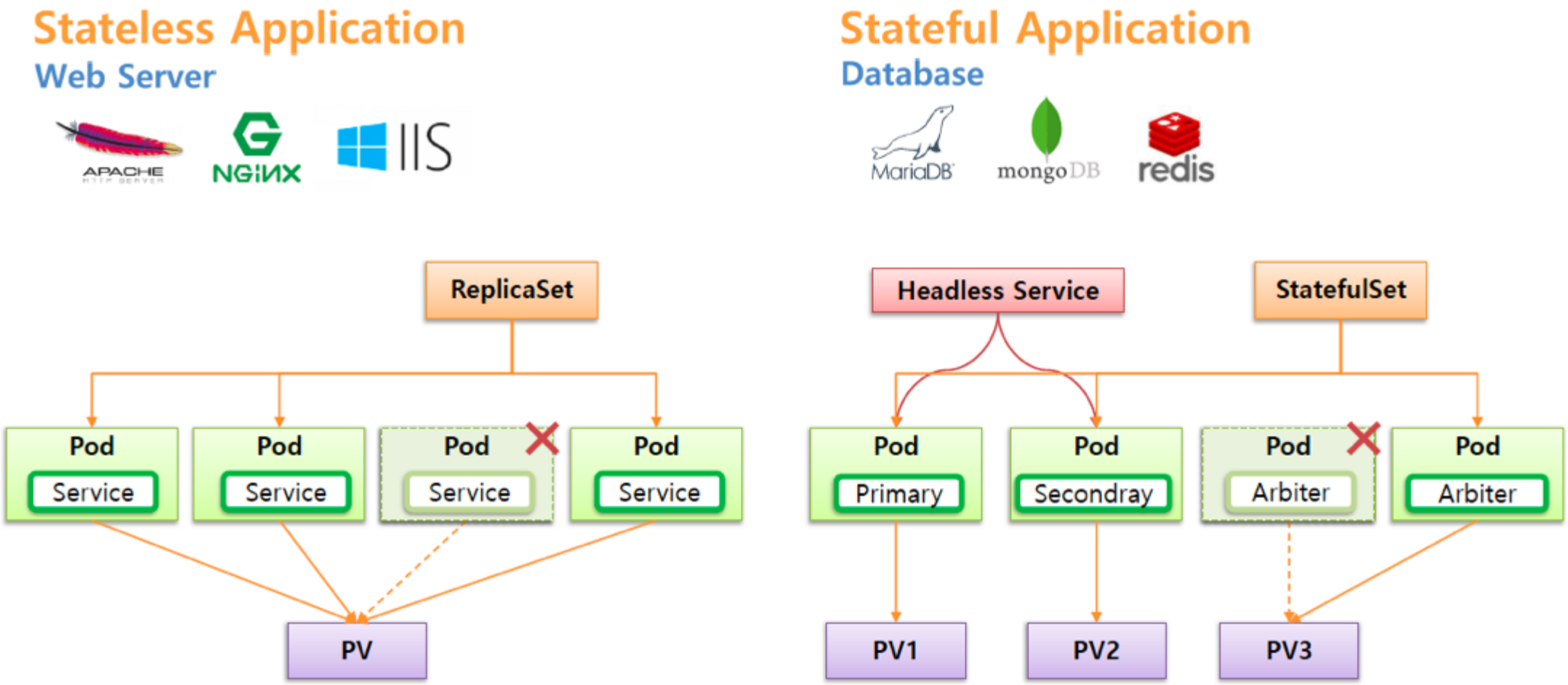
You're defining a readiness probe that will be executed every second.

The readiness probe will perform an HTTP GET request against our container.

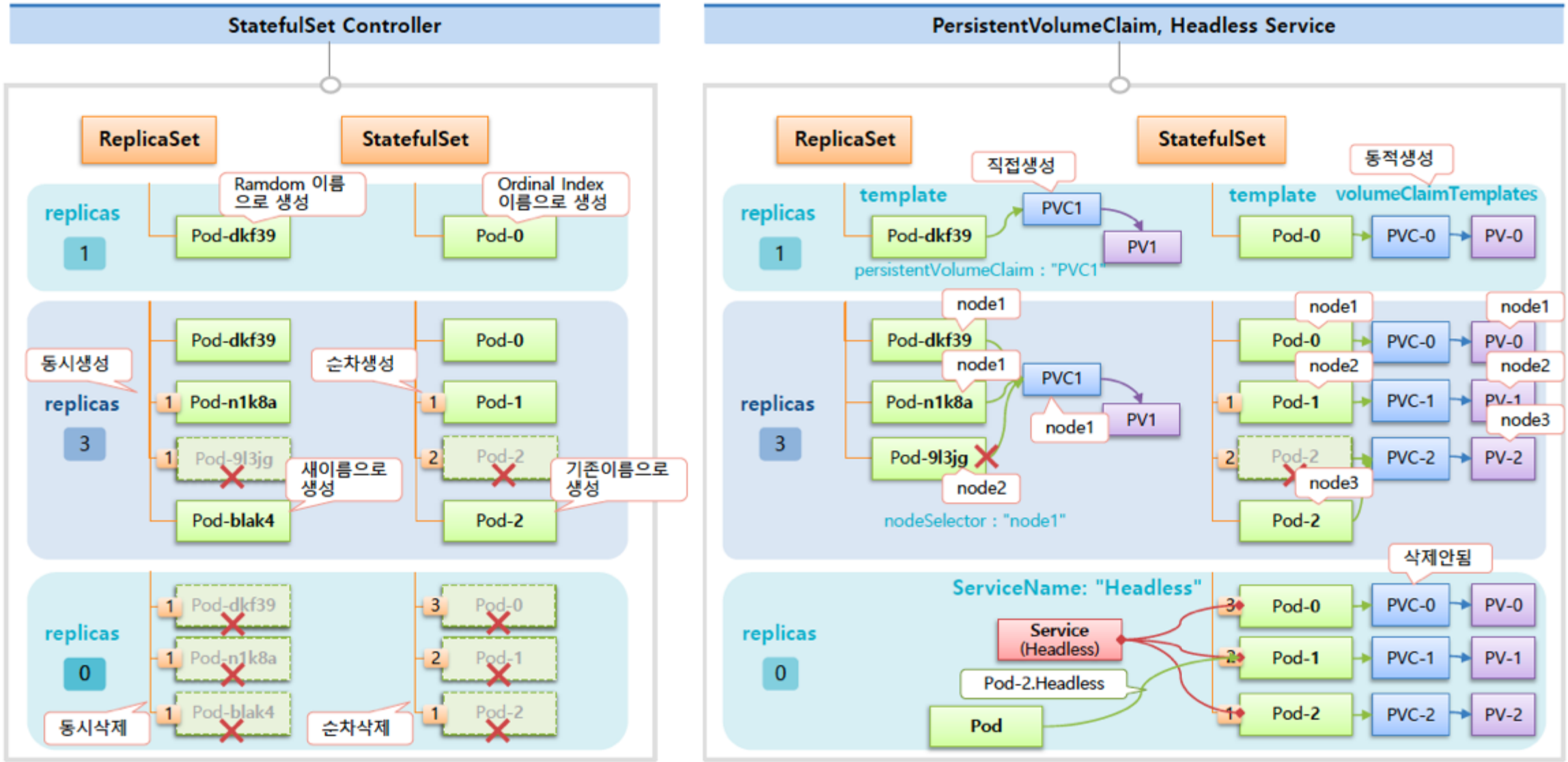
StatefulSet



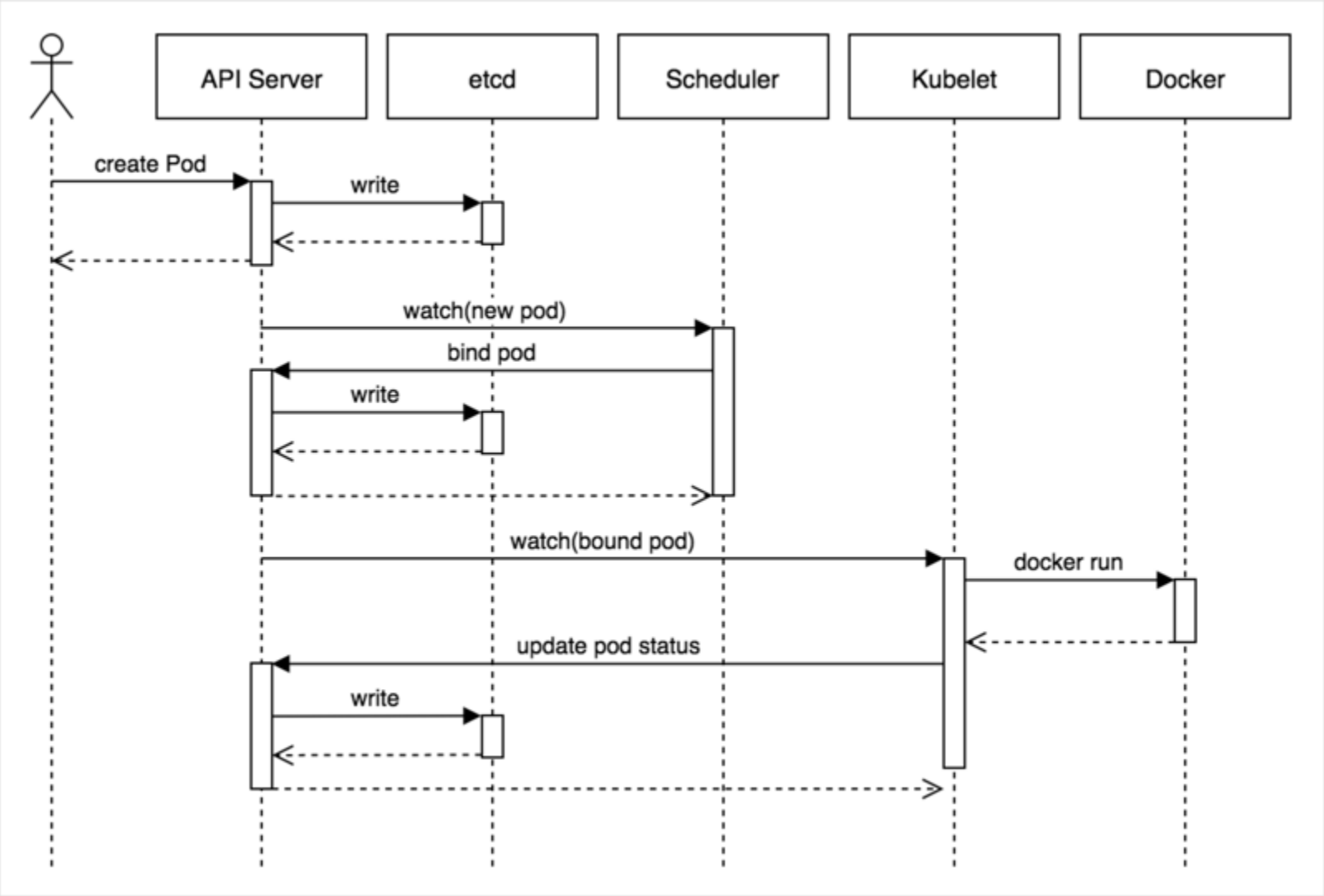
StatefulSet



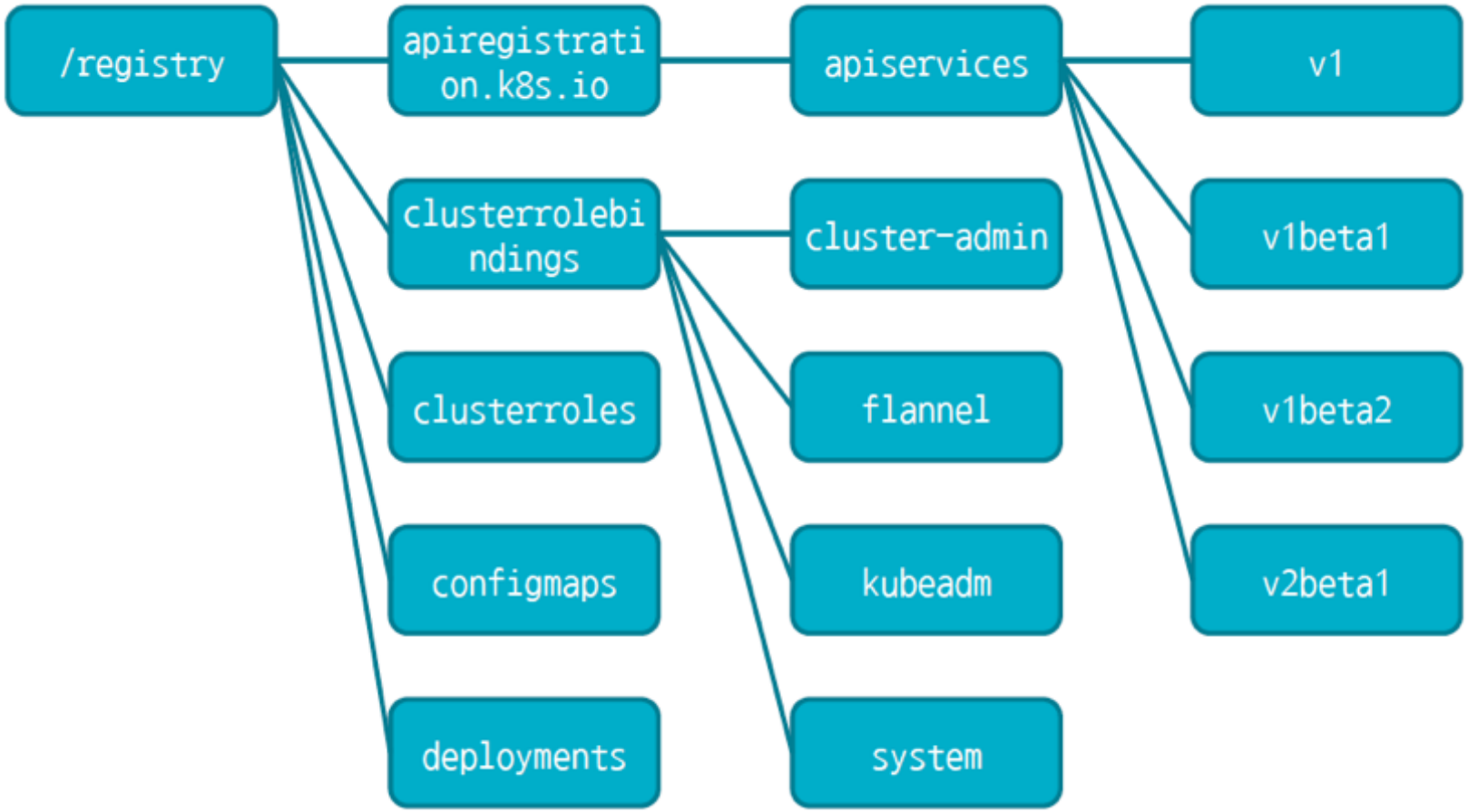
StatefulSet



etcd: persist data



etcd Structure

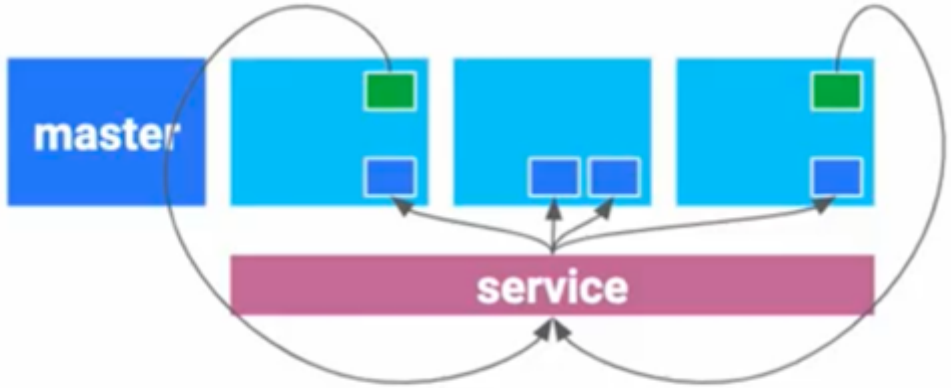


kube-dns

kube-dns is an addon that resolves `$service.$namespace(.svc.cluster.local)` to the right Service IP. This way you can talk to other apps by knowing only the Service name.

Runs as a deployment fronted by a service

When deployed, every container uses the service ip for kube-dns in its /etc/resolv.conf



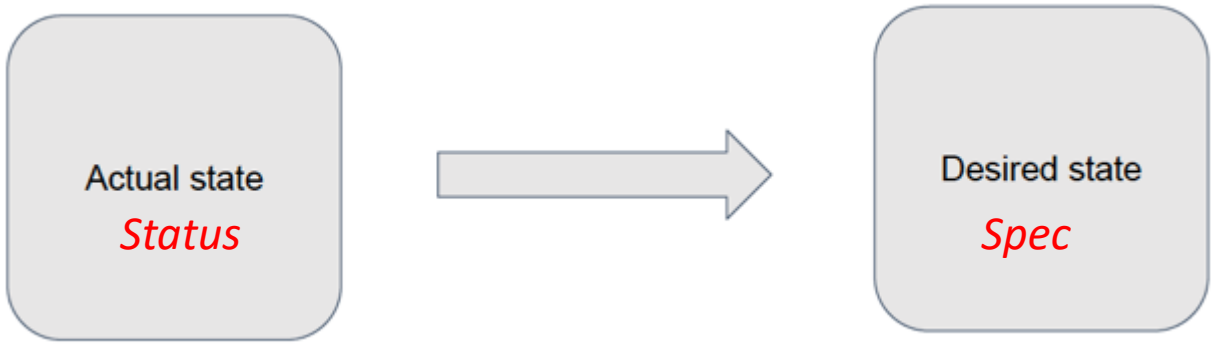
Declarative Configuration

Before:

- **You:** provide exact set of instructions to drive to desired state
- **System:** executes instructions
- **You:** monitor system, and provide further instructions if it deviates.

After:

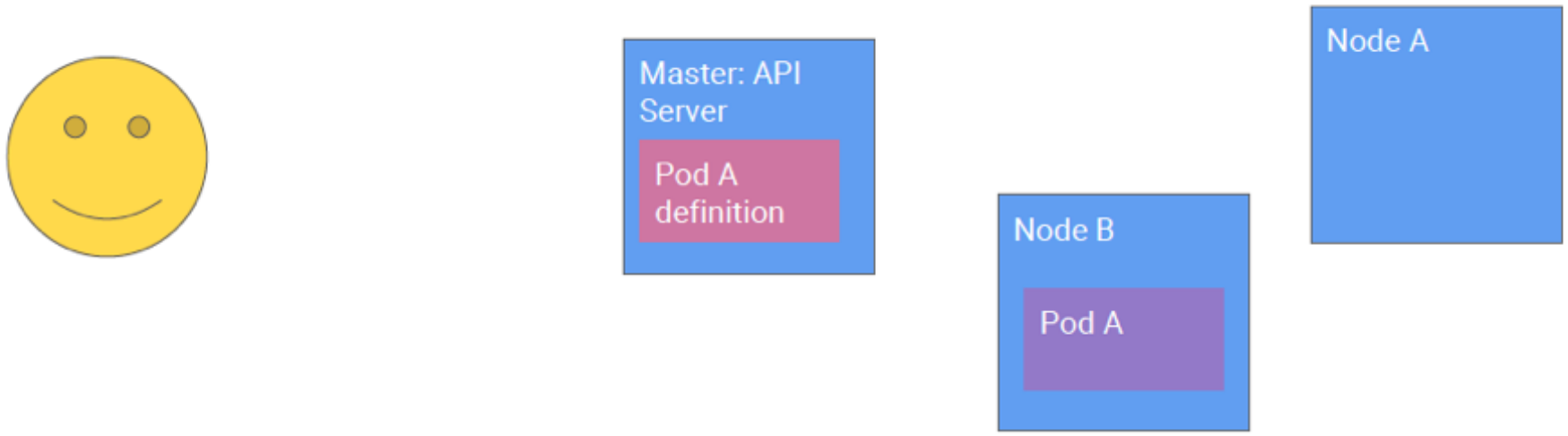
- **You:** define desired state
- **System:** works to drive towards that state



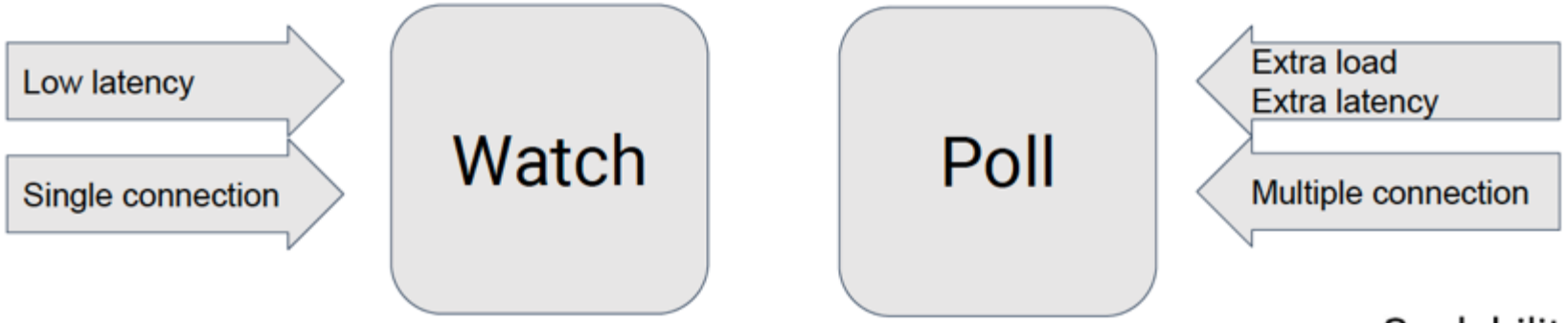
Kubernetes Way to Deploy Workload

The Kubernetes way!

- **You:** create API object that is persisted on kube API server until deletion
- **System:** all components work in parallel to drive to that state

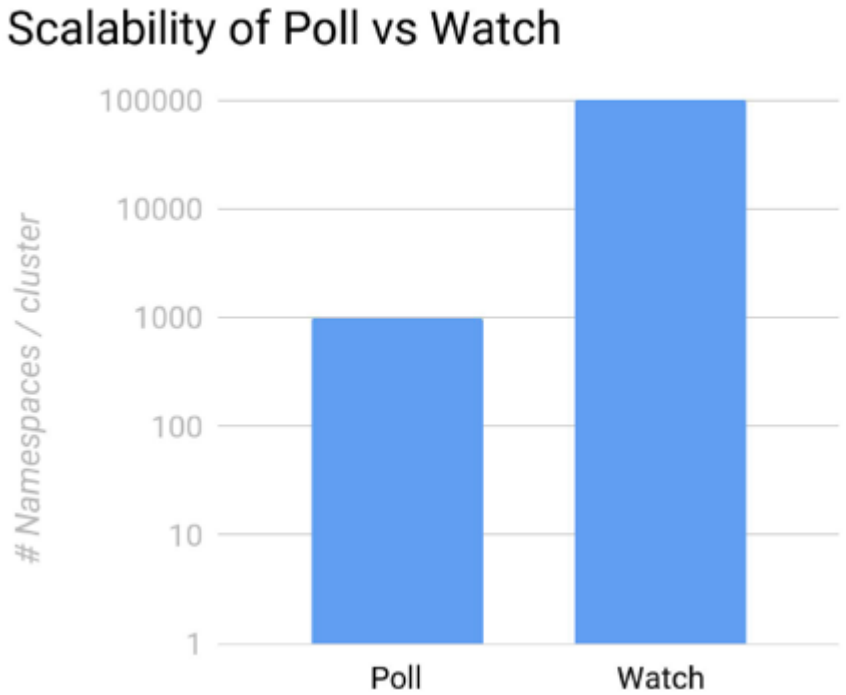


Kubernetes API server: Watch vs Poll

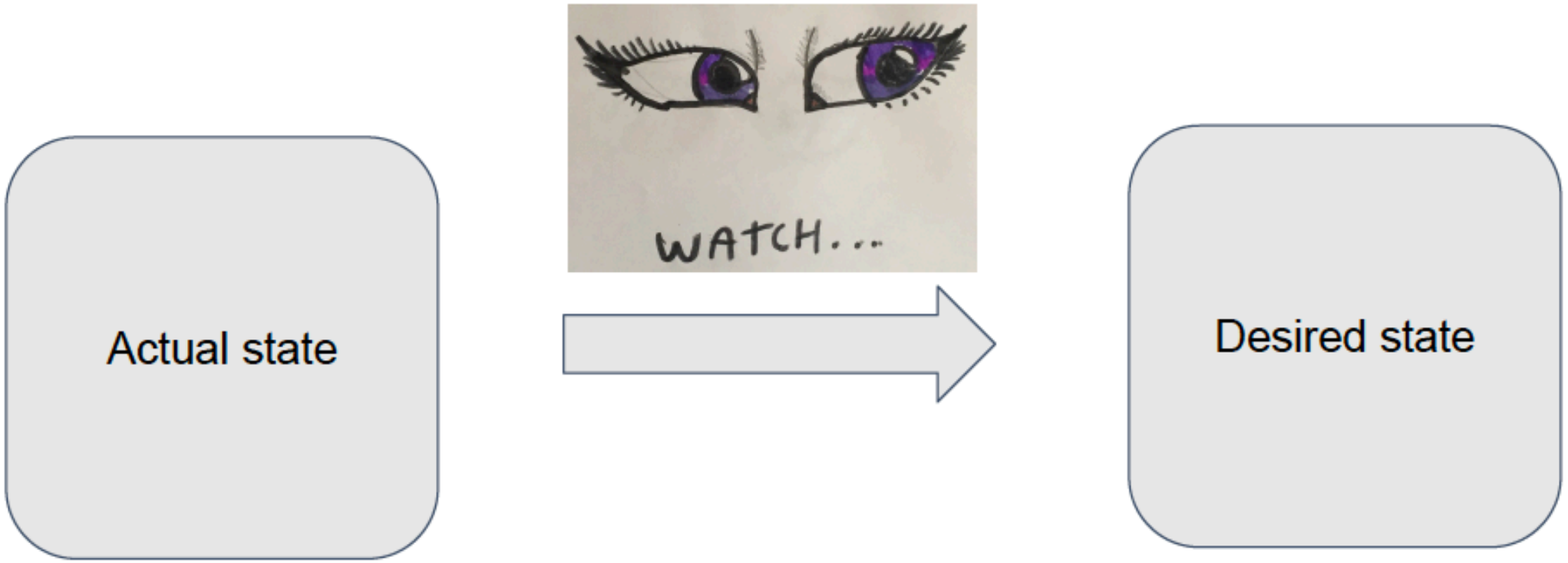


Kubelet on nodes:

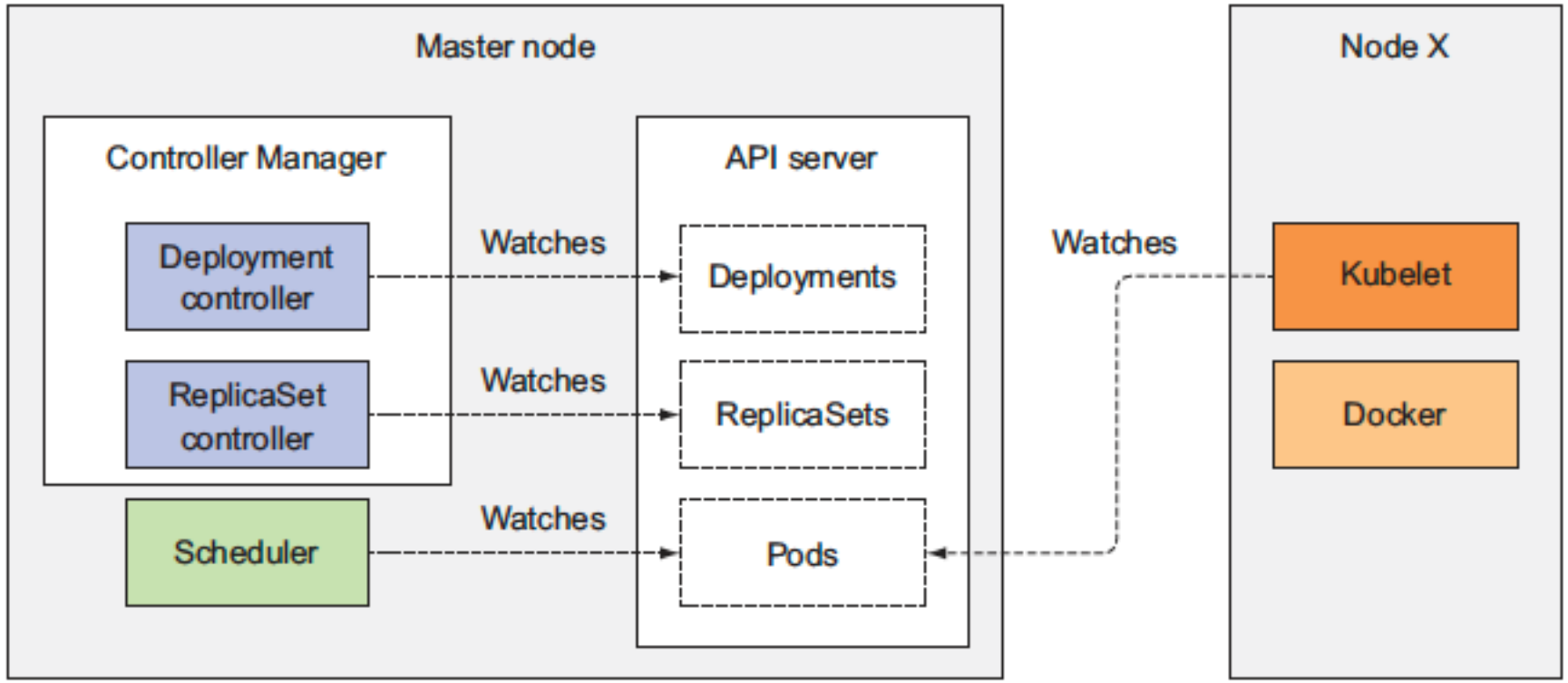
- Previous: periodically poll kube-apiserver for secrets and configmaps
- Now: watch individual secrets
- OSS PR: Kubelet watches necessary secrets/configmaps instead of periodic polling #64752



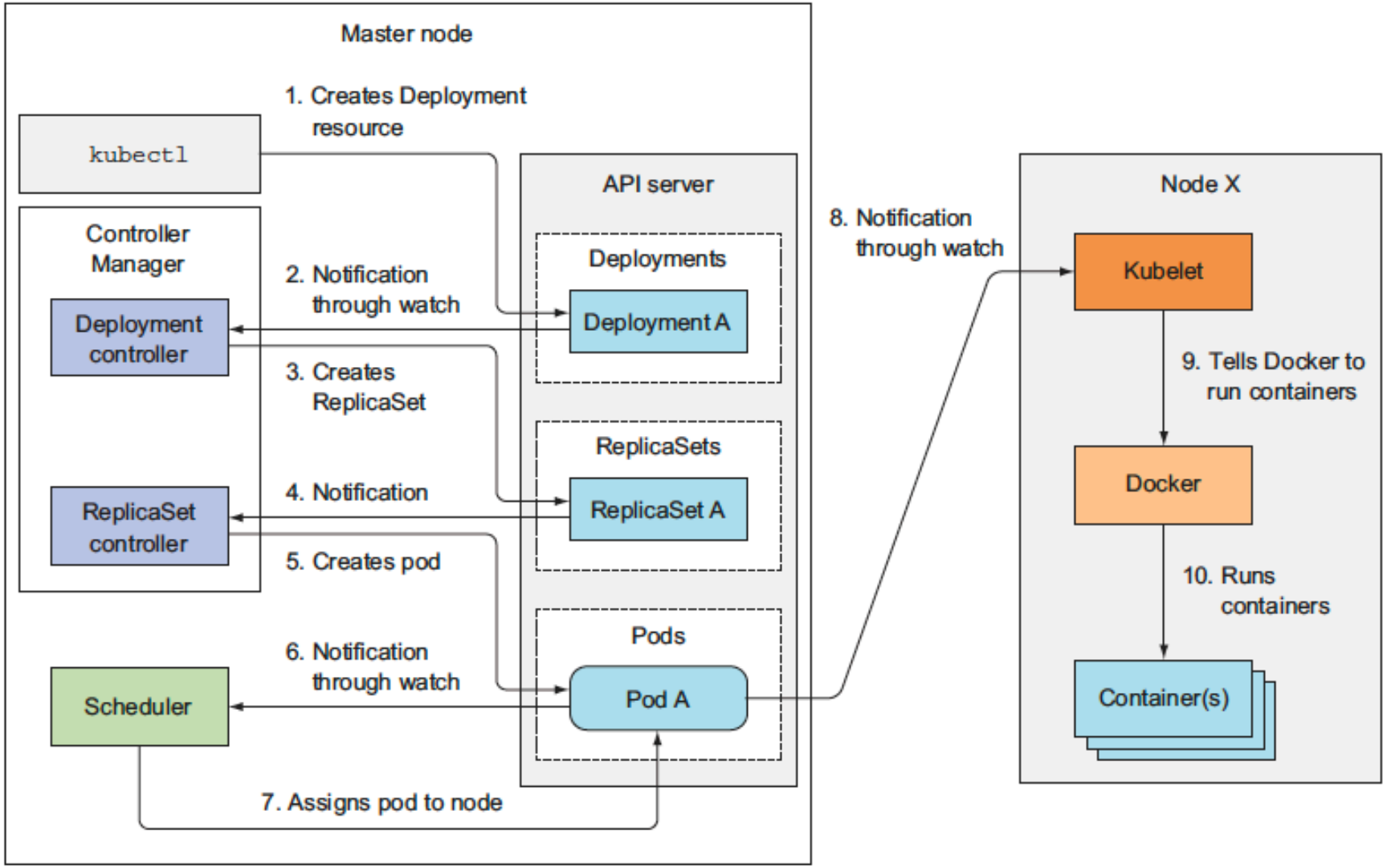
Watch Event



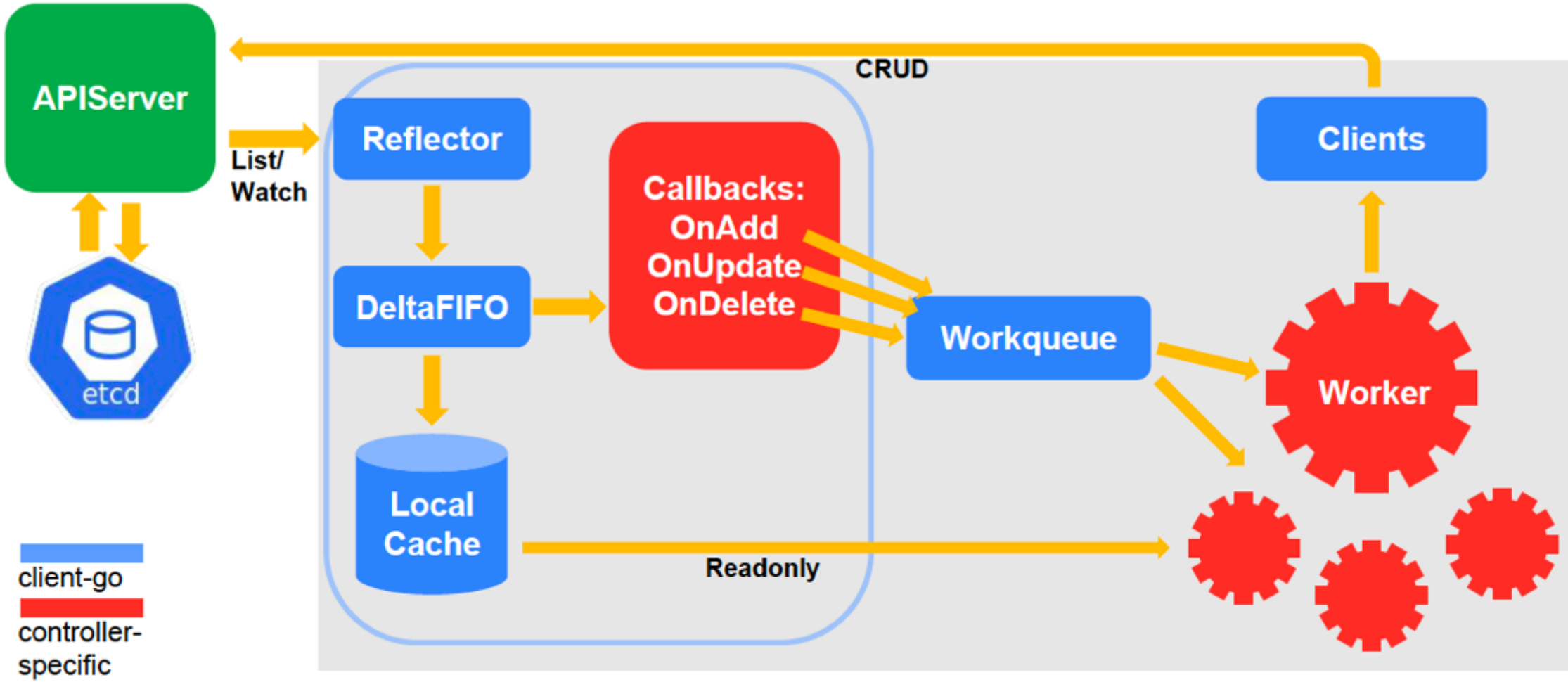
Watches

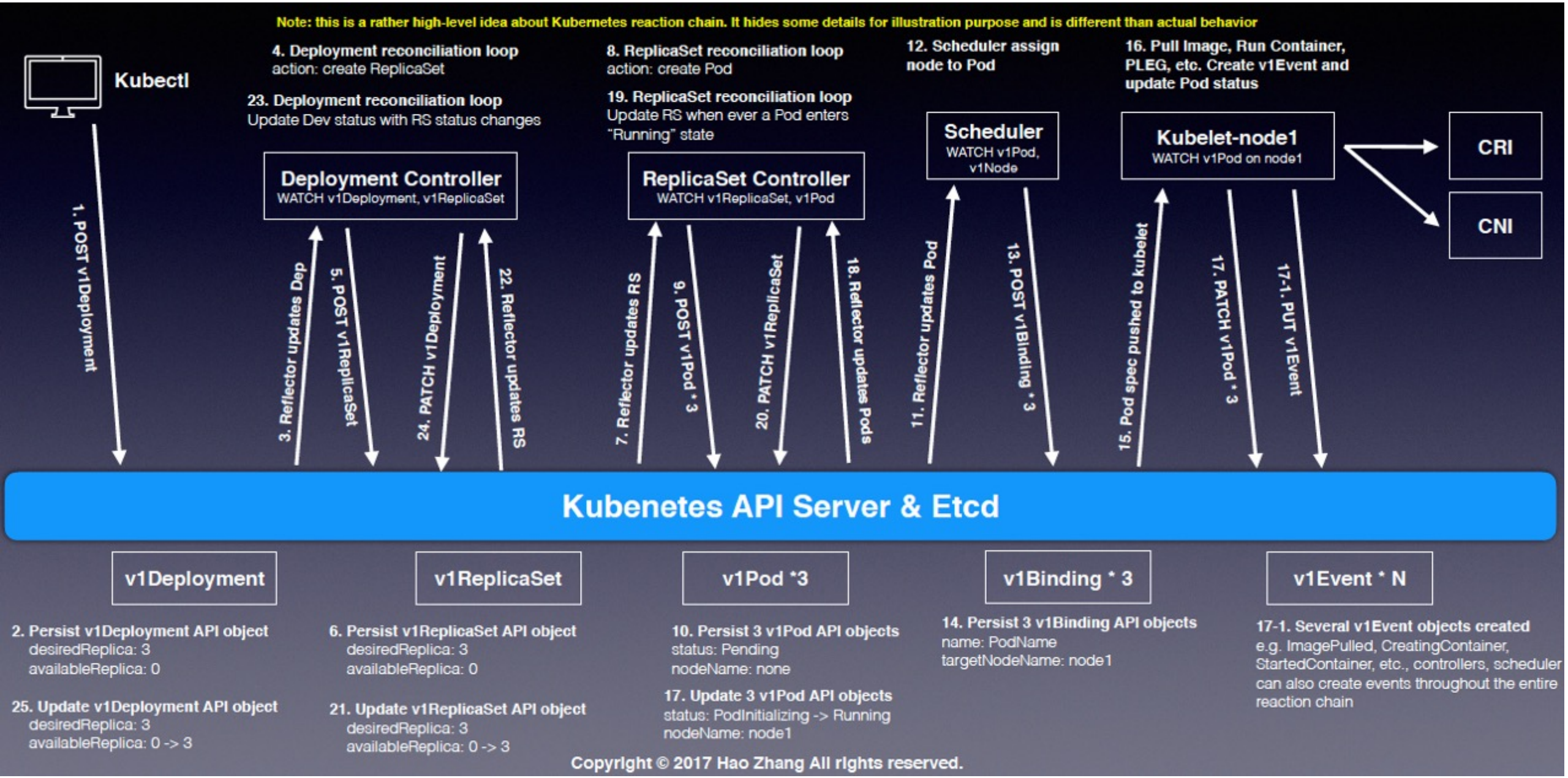

Master node
Controller Manager
Deployment controller
ReplicaSet controller
Scheduler
API server
Deployments
ReplicaSets
Pods
Watches
Watches
Watches
Watches
Node X
Kubelet
Docker

Event Chain

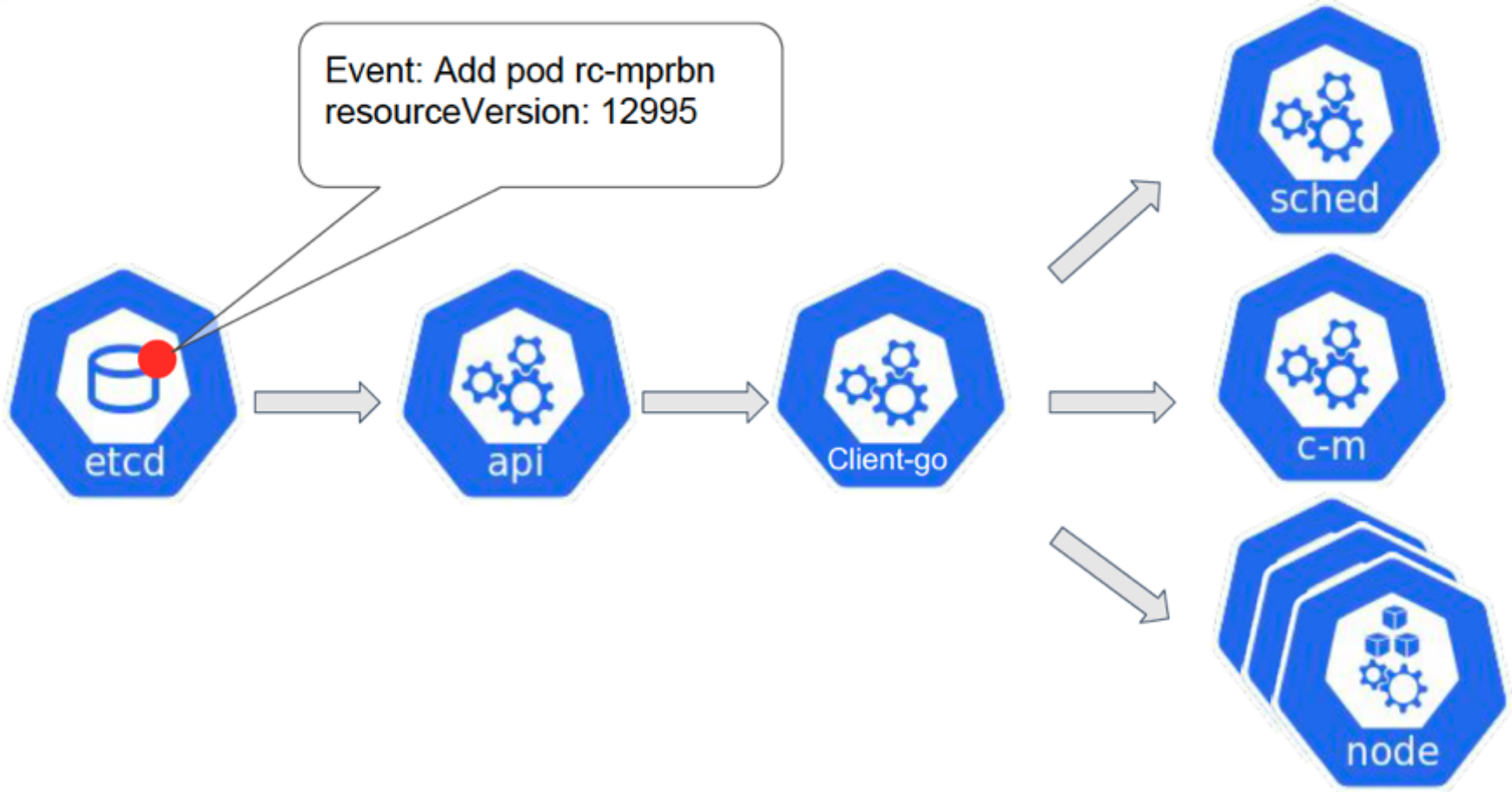

Master node
1. Creates Deployment resource
kubectl
API server
Controller Manager
Deployments
2. Notification through watch
Deployment controller
Deployment A
3. Creates ReplicaSet
ReplicaSets
4. Notification
ReplicaSet controller
ReplicaSet A
5. Creates pod
Pods
6. Notification through watch
Scheduler
Pod A
7. Assigns pod to node
8. Notification through watch
Node X
Kubelet
9. Tells Docker to run containers
Docker
10. Runs containers
Container(s)

Kubernetes Controller Workflow

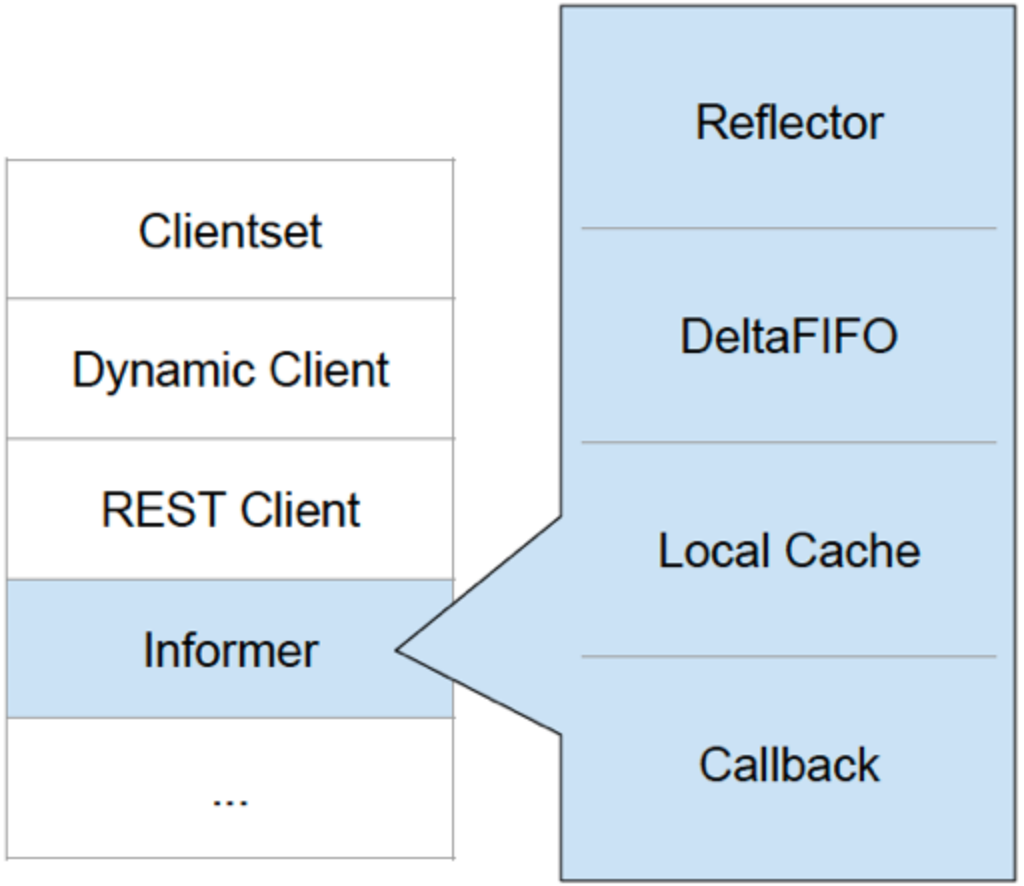




Client-go (1)

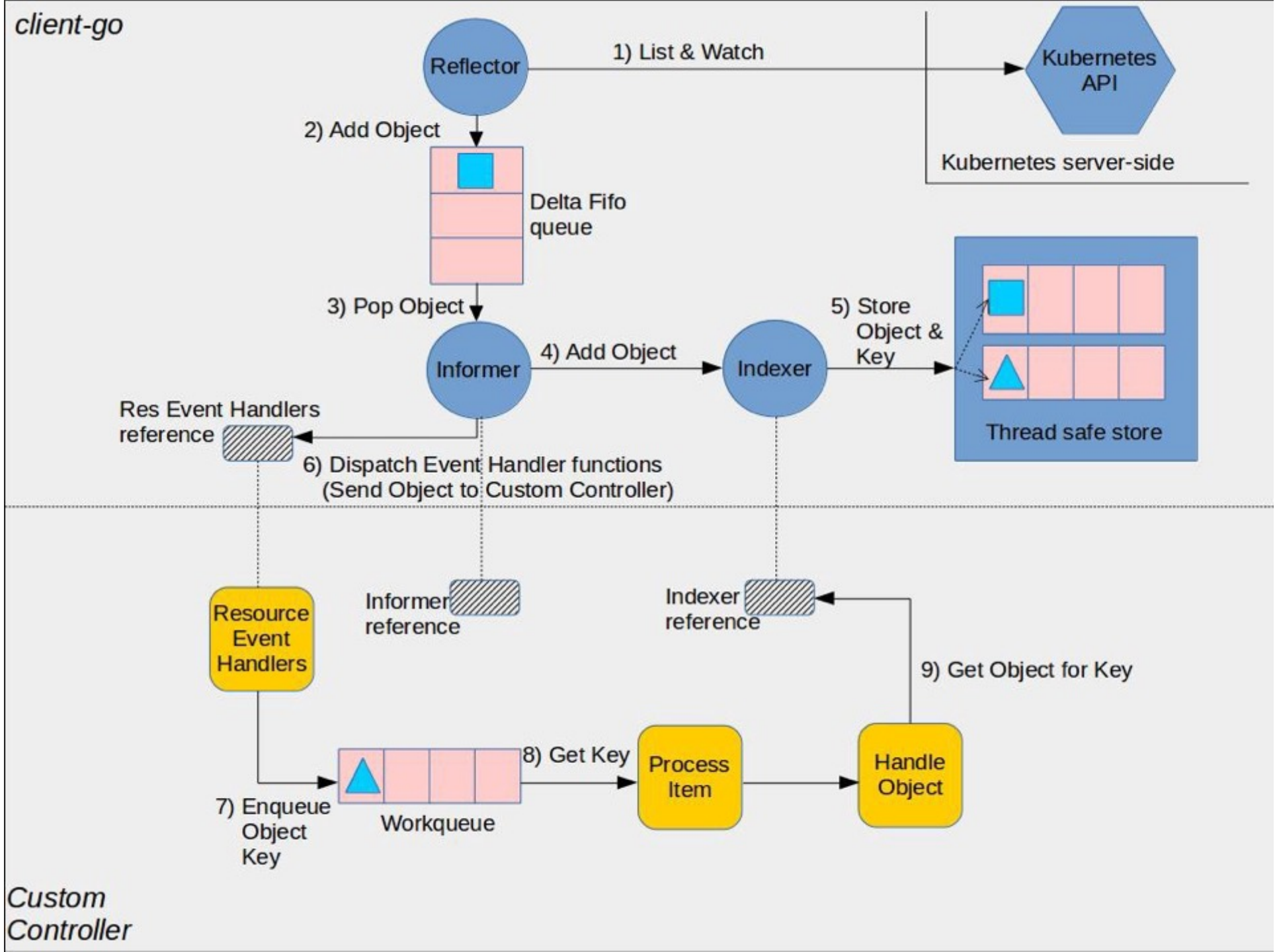

Event: Add pod rc-mprbn
resourceVersion: 12995
etcd
api
Client-go
sched
c-m
node

Client-go (2)

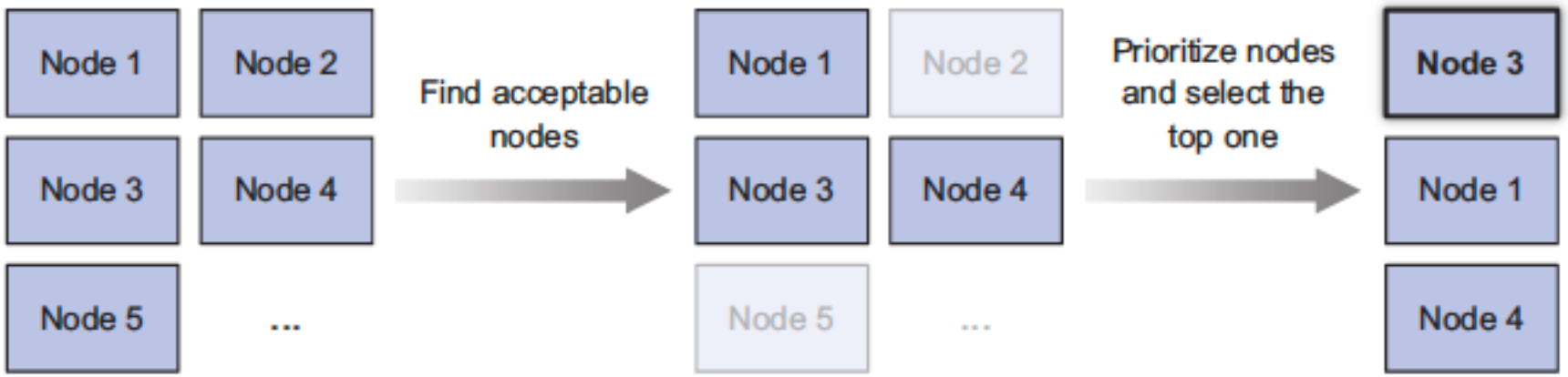


k8s.io/client-go/tools/cache
k8s.io/client-go/informers

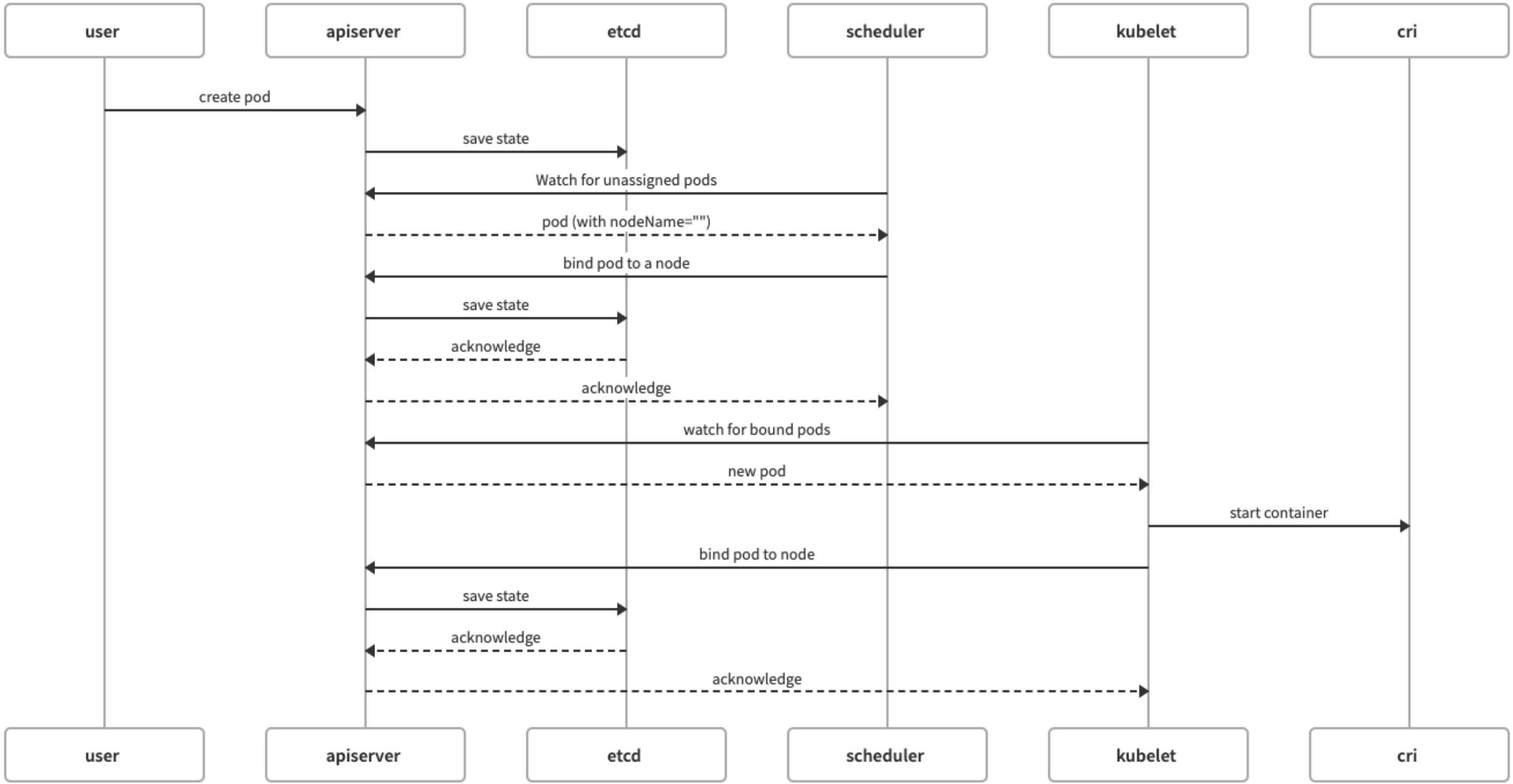
- Useful component for building event-oriented controllers
- Used by control plane controllers, kubelet, etc.
- Reflector used by kube-apiserver watch cache


client-go
Reflector
1) List & Watch
Kubernetes API
Kubernetes server-side
2) Add Object
Delta Fifo queue
3) Pop Object
Informer
4) Add Object
Indexer
5) Store Object & Key
Thread safe store
Res Event Handlers reference
6) Dispatch Event (Send Object to Handler functions Custom Controller)
Resource Event Handlers
Informer reference
Indexer reference
9) Get Object for Key
8) Get Key
Process Item
Handle Object
7) Enqueue Object Key
Workqueue
Custom Controller

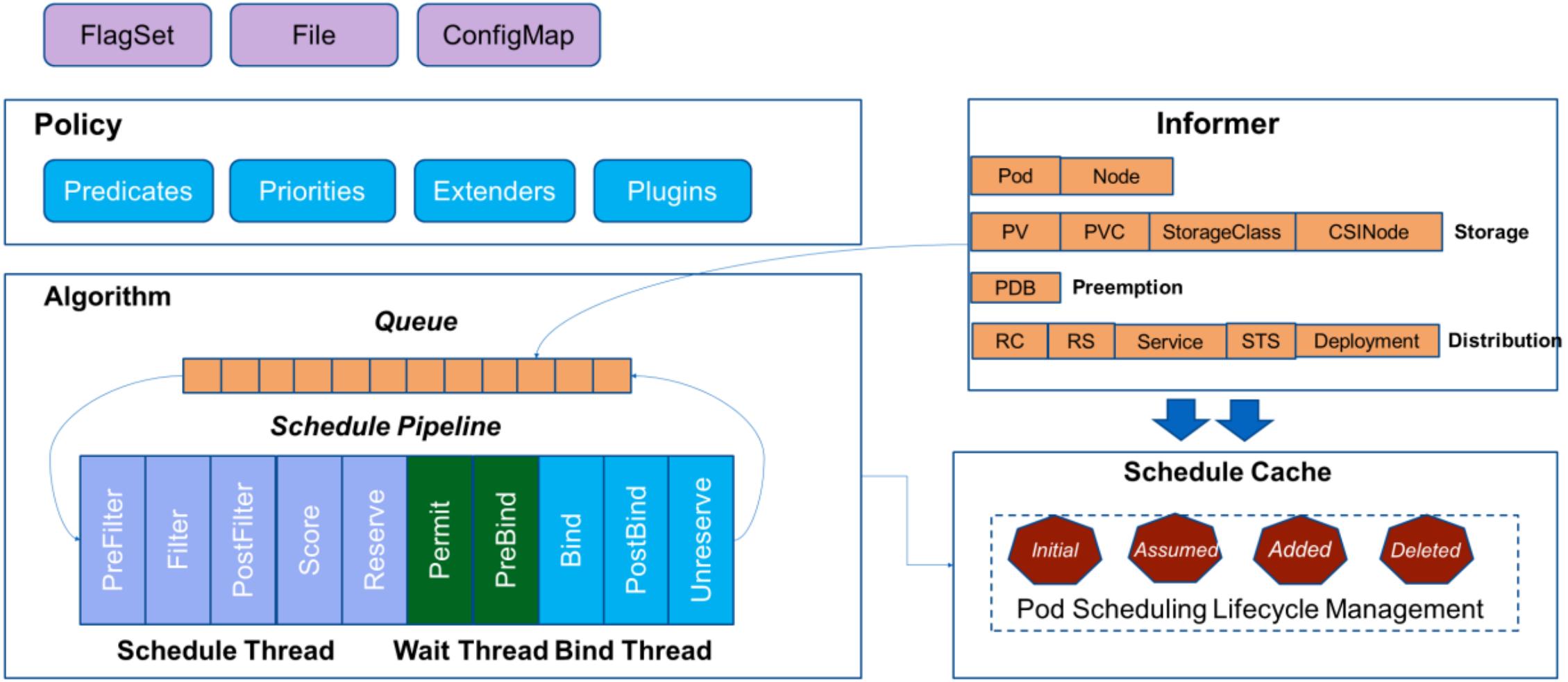
Kubernetes Scheduler



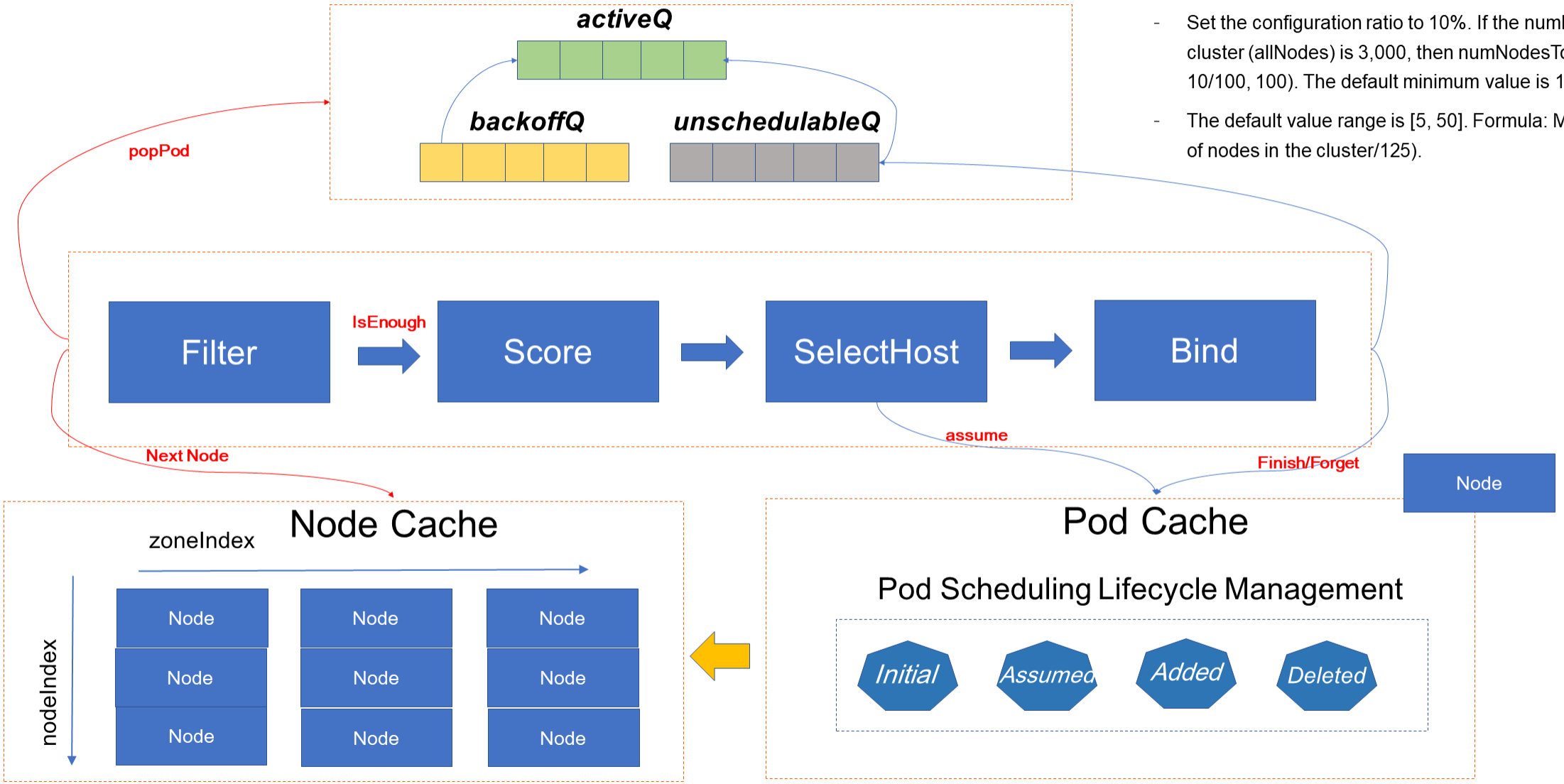
Kubernetes Scheduler



Scheduler Overview



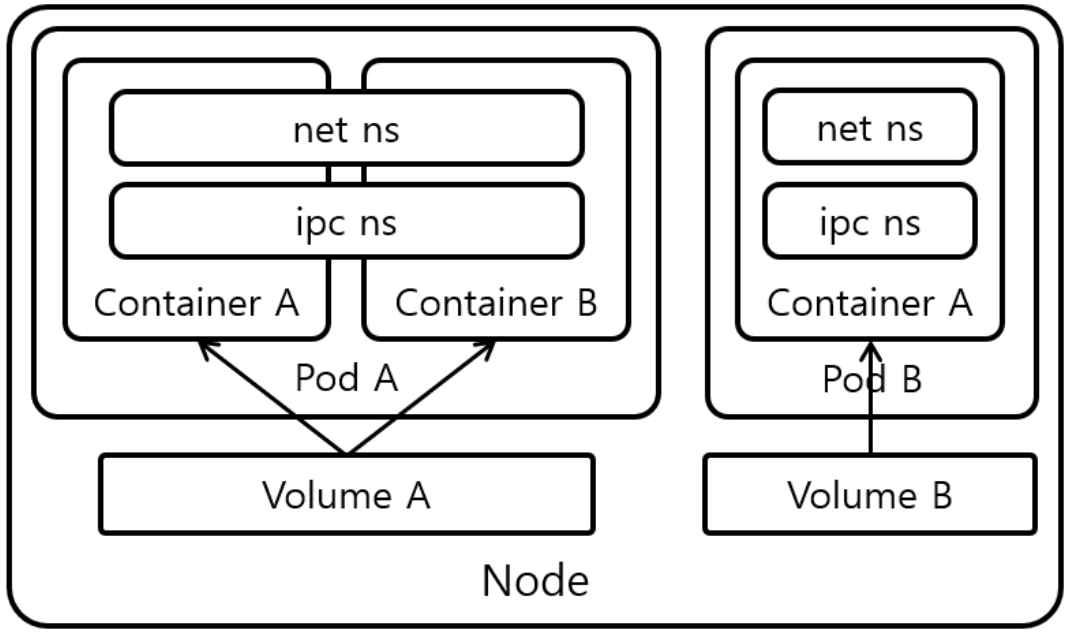
Scheduling Process



Configure the node sampling scope

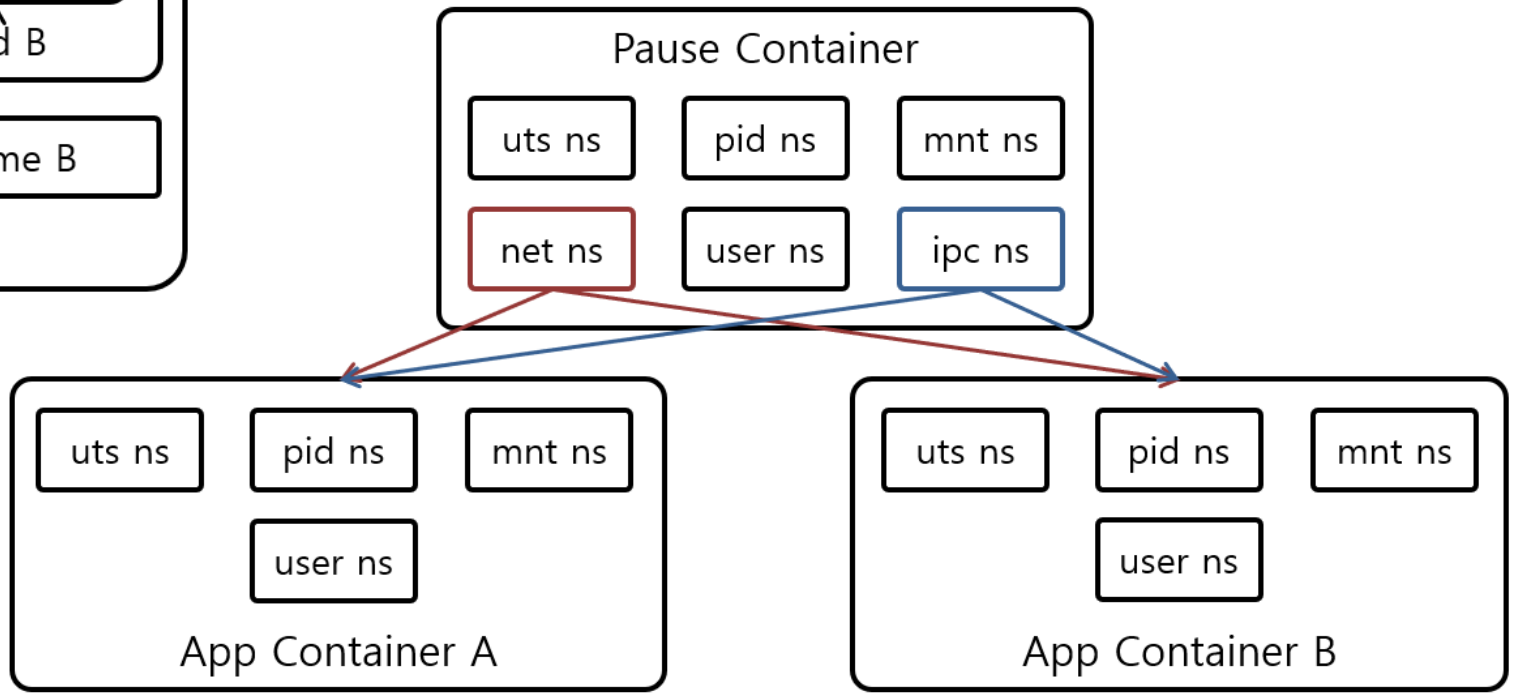
- Set the configuration ratio to 10%. If the number of nodes in the cluster (allNodes) is 3,000, then numNodesToFind = Max(3000 x 10/100, 100). The default minimum value is 100.
- The default value range is [5, 50]. Formula: Max(5, 50 - Number of nodes in the cluster/125).

Pause Container

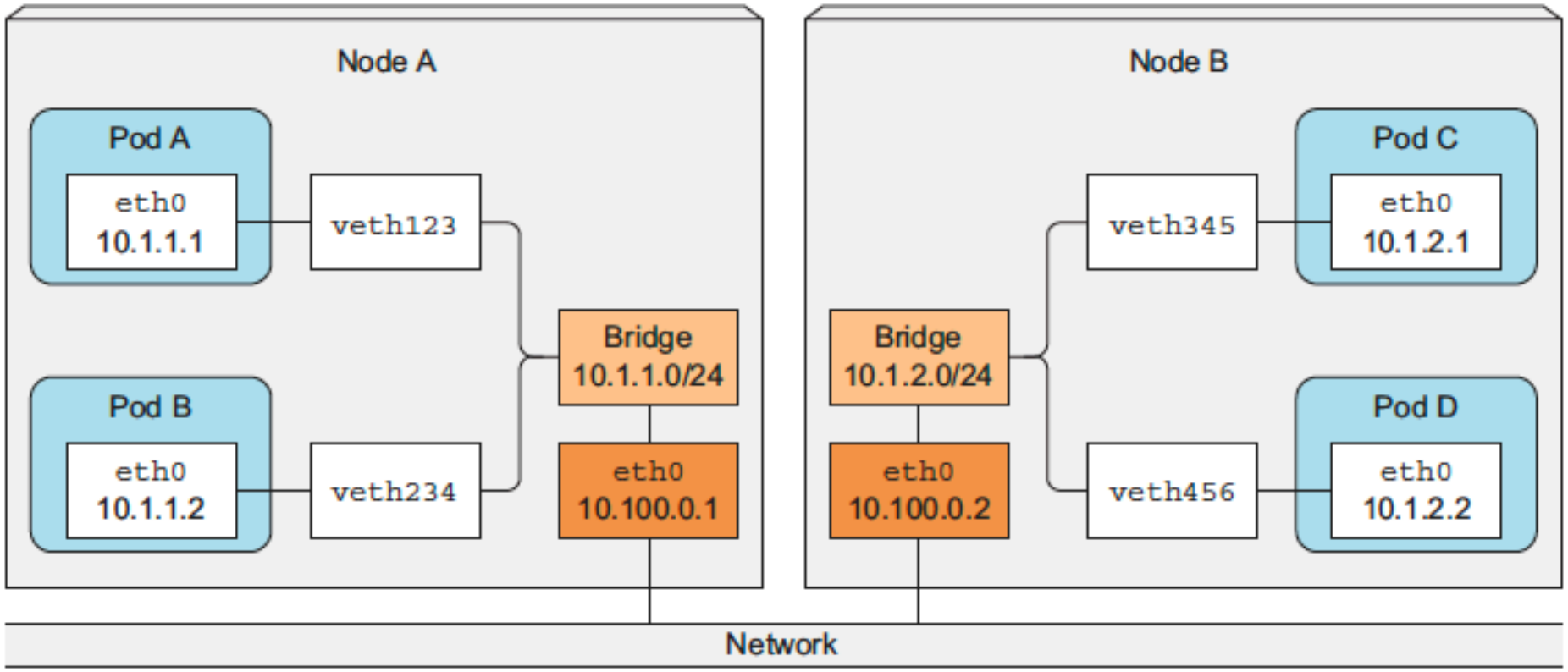


Init Process killed ->SIGKILL to other cloned processes

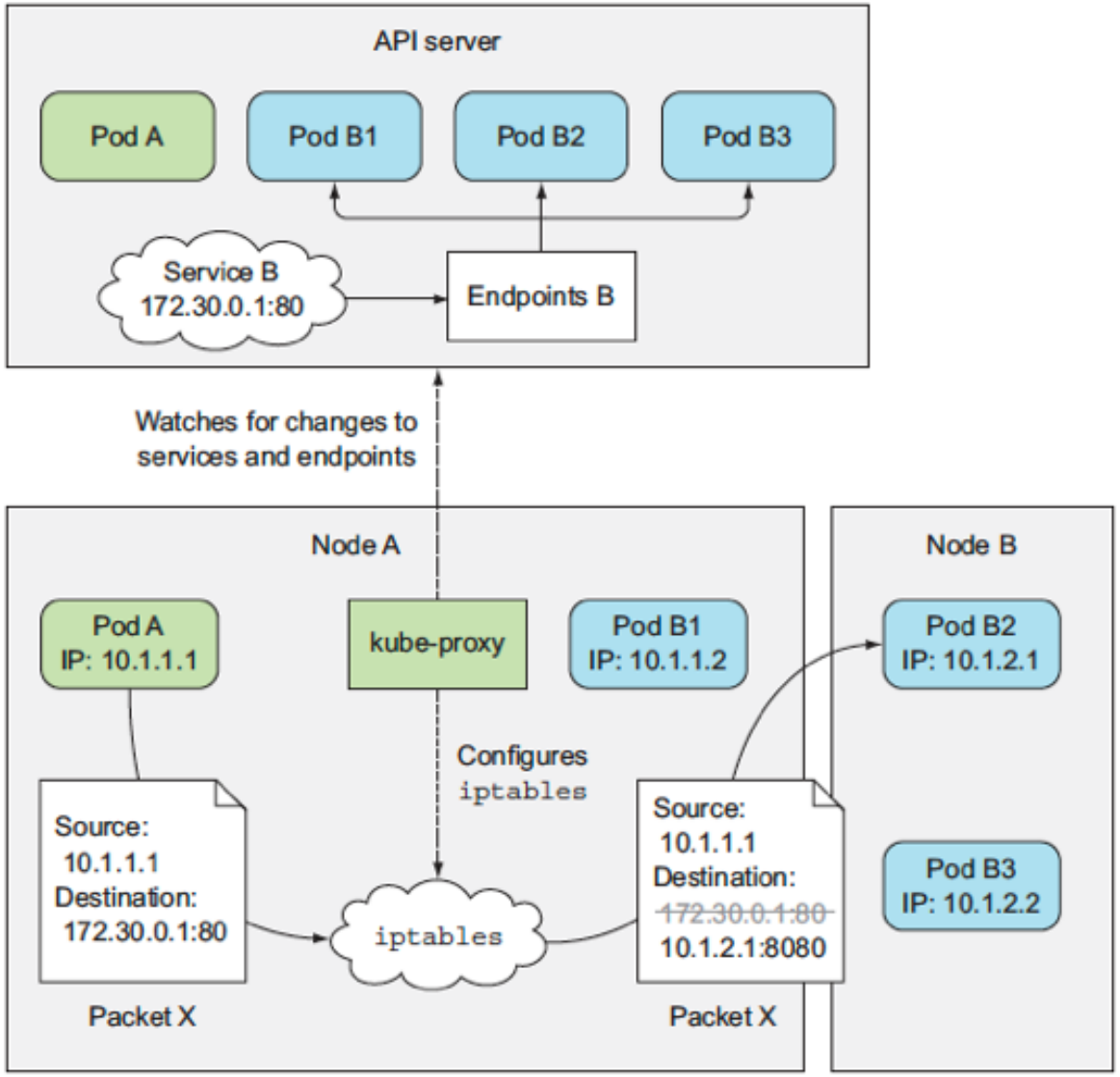
Pause binary Init Process: **pause()** syscall -> blocking until SIG

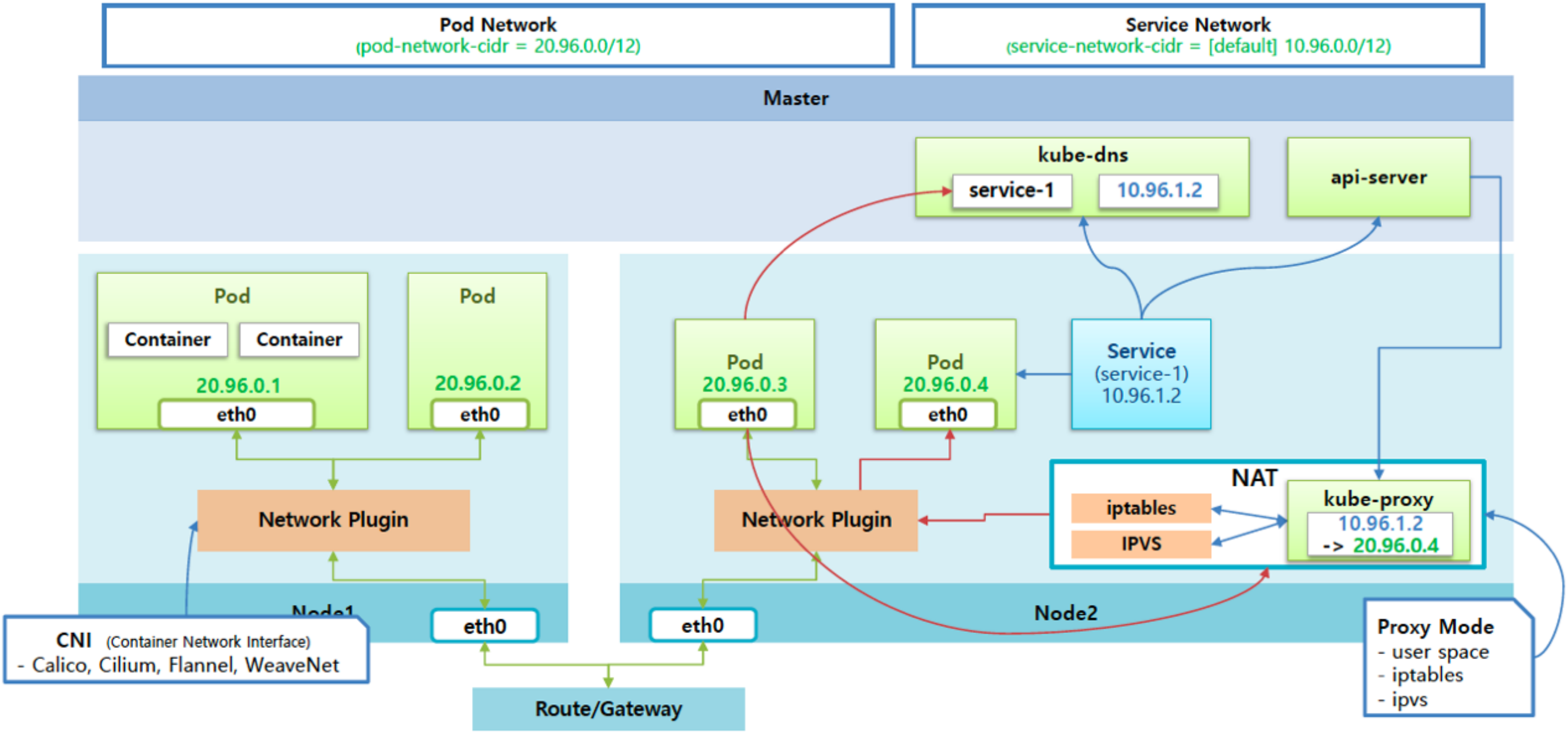


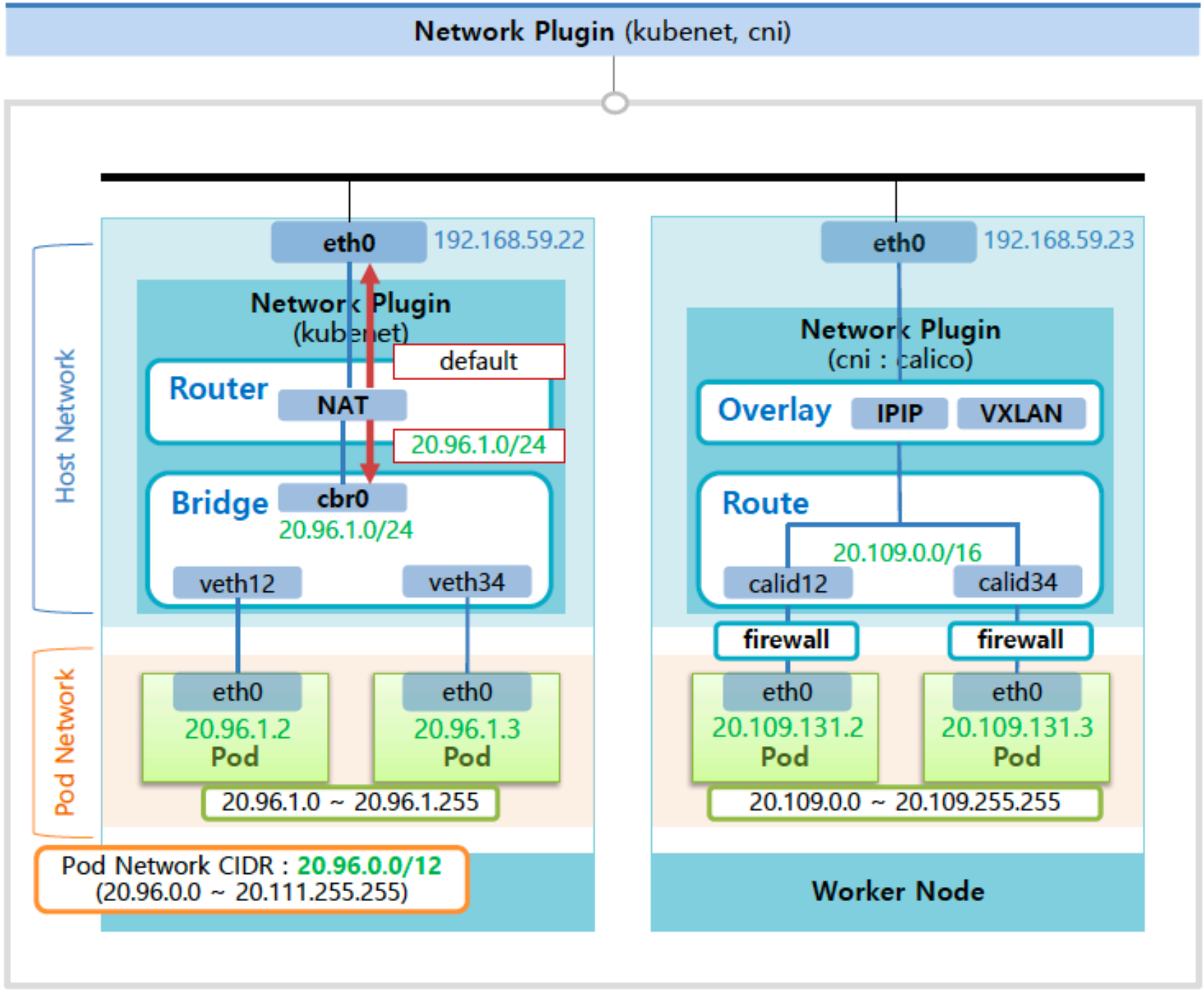
Communication between Pods and Nodes

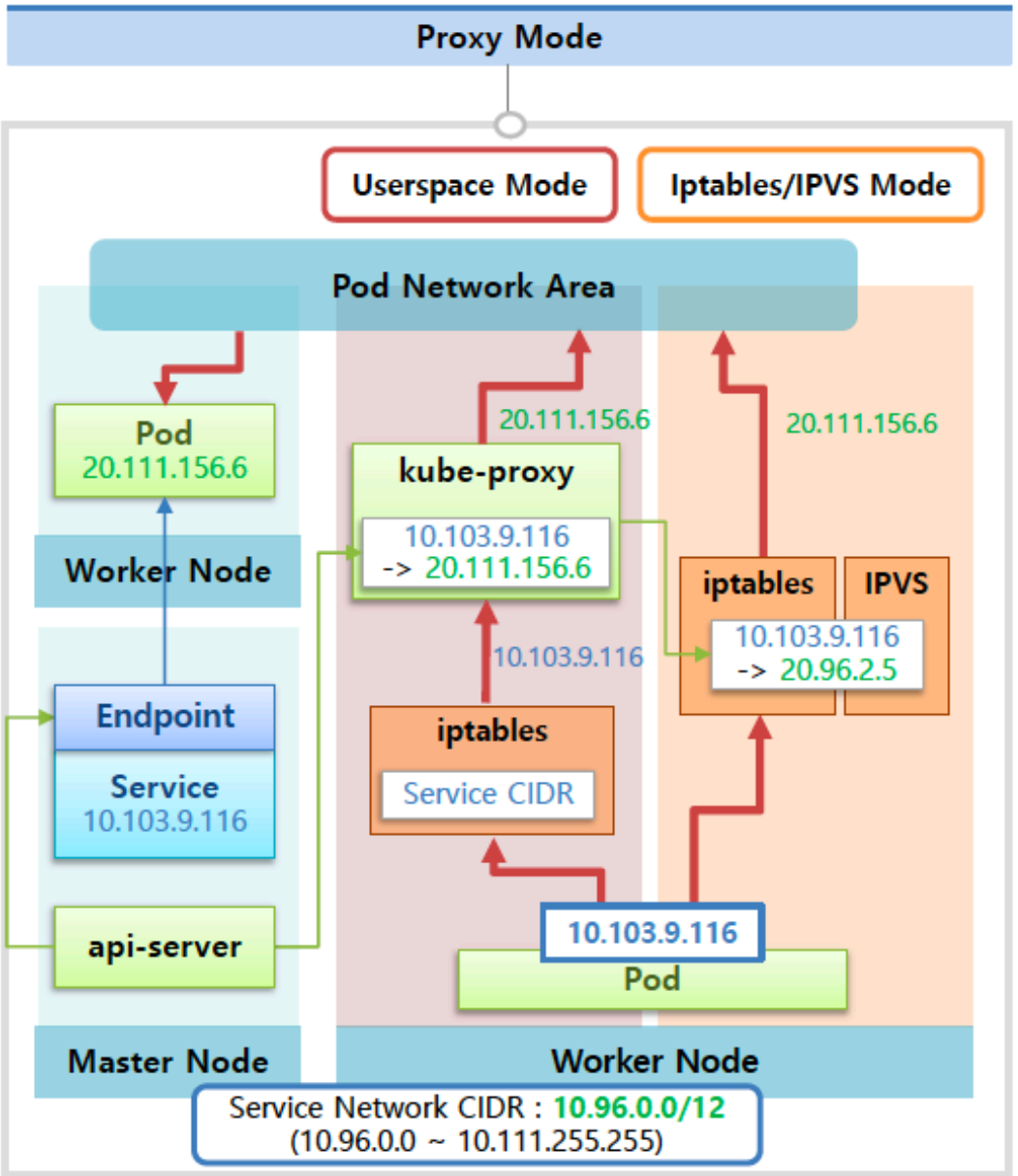


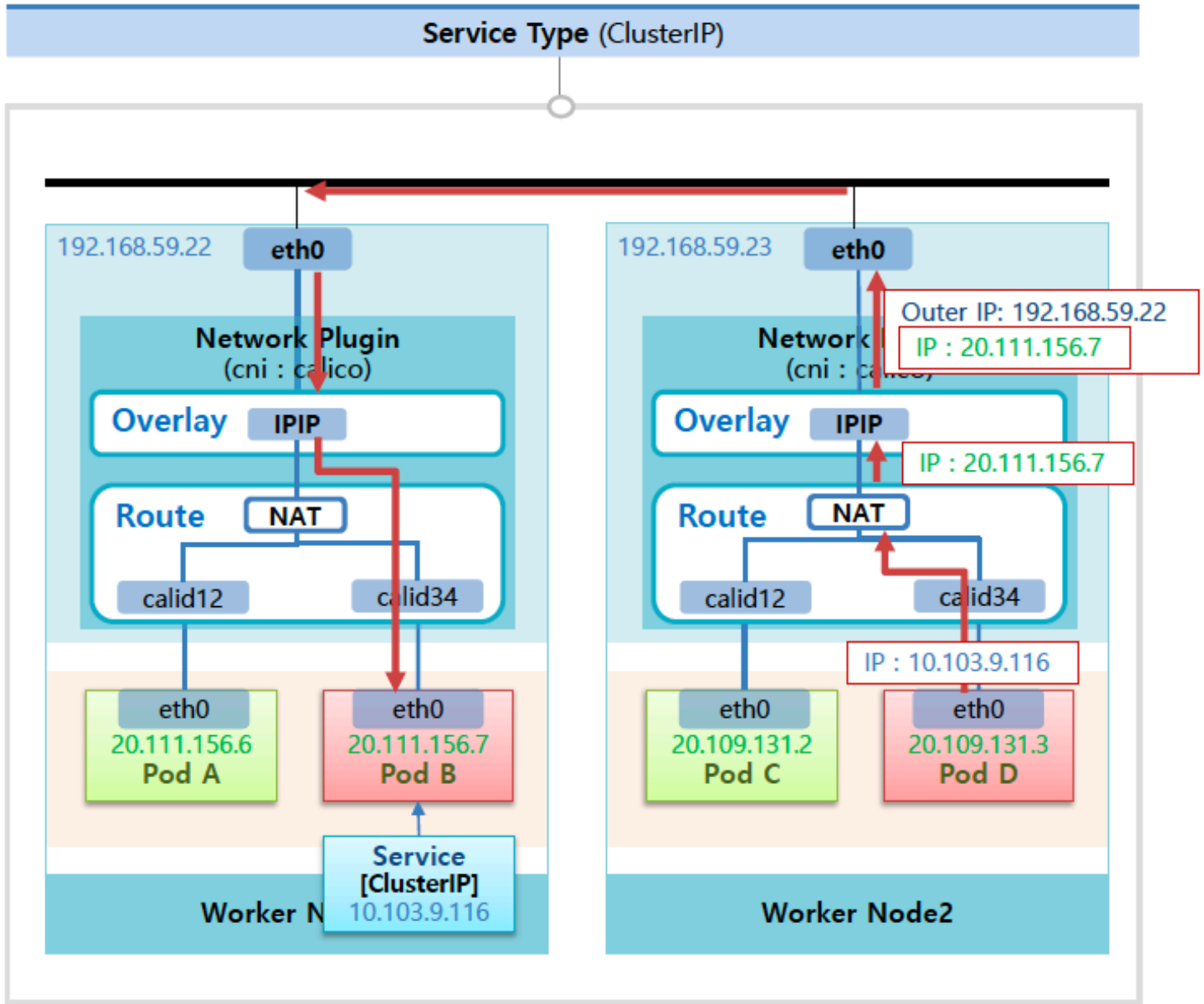
kube-proxy & iptables

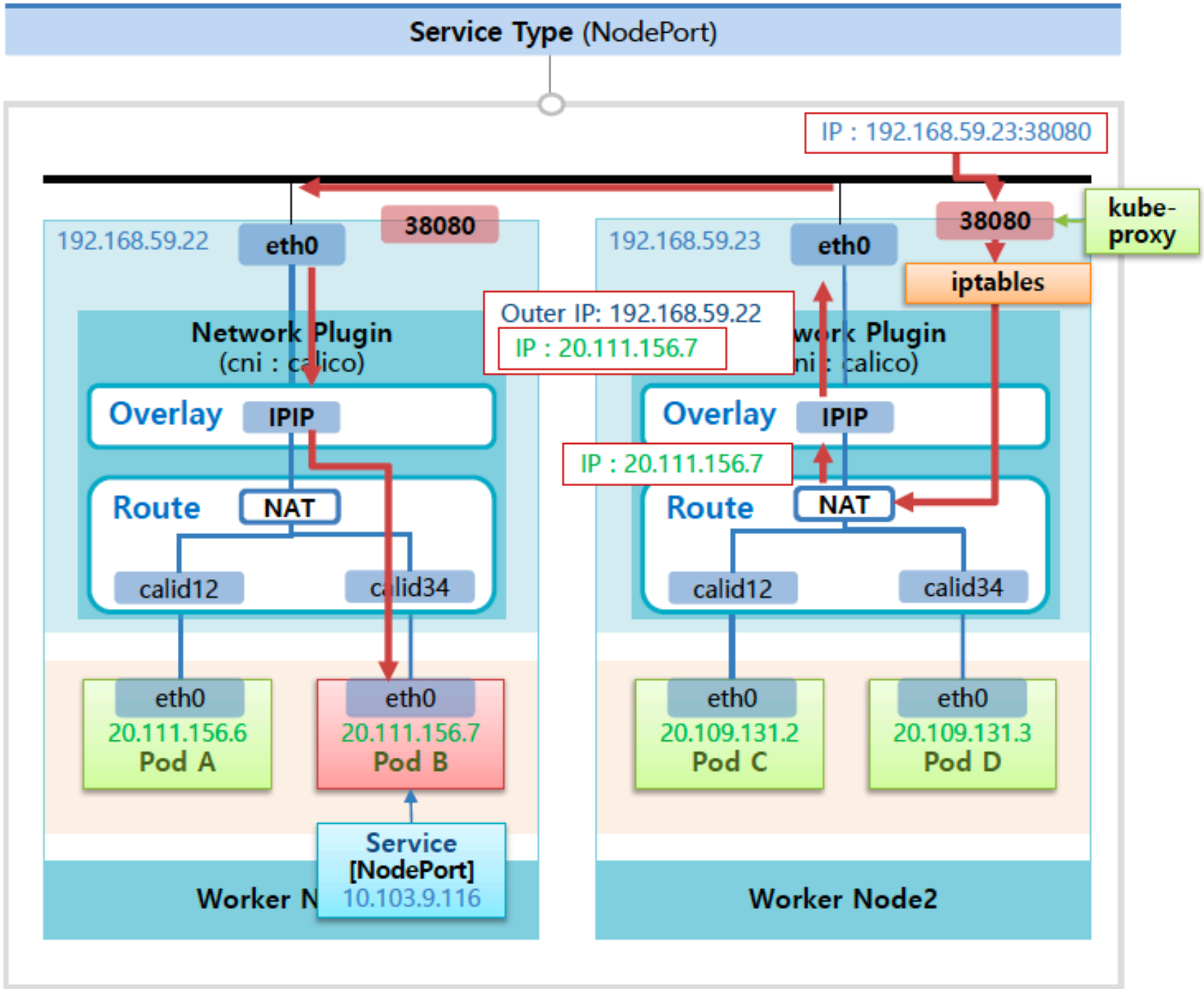



Pod Network
(pod-network-cidr = 20.96.0.0/12)
Service Network
(service-network-cidr = [default] 10.96.0.0/12)
Master
kube-dns
service-1
10.96.1.2
api-server
Pod
Container
Container
20.96.0.1
eth0
Pod
20.96.0.2
eth0
Pod
20.96.0.3
eth0
Pod
20.96.0.4
eth0
Service
(service-1)
10.96.1.2
NAT
iptables
IPVS
kube-proxy
10.96.1.2
-> 20.96.0.4
Network Plugin
Network Plugin
Node1
Node2
eth0
eth0
CNI (Container Network Interface)
- Calico, Cilium, Flannel, WeaveNet
Route/Gateway
Proxy Mode
- user space
- iptables
- ipvs


Network Plugin (kubenet, cni)
Host Network
Pod Network
eth0
192.168.59.22
Network Plugin
(kubenet)
default
Router
NAT
20.96.1.0/24
Bridge
cbr0
20.96.1.0/24
veth12
veth34
eth0
20.96.1.2
Pod
eth0
20.96.1.3
Pod
20.96.1.0 ~ 20.96.1.255
Pod Network CIDR : 20.96.0.0/12
(20.96.0.0 ~ 20.111.255.255)
eth0
192.168.59.23
Network Plugin
(cni : calico)
Overlay
IPIP
VXLAN
Route
20.109.0.0/16
calid12
calid34
firewall
firewall
eth0
20.109.131.2
Pod
eth0
20.109.131.3
Pod
20.109.0.0 ~ 20.109.255.255
Worker Node

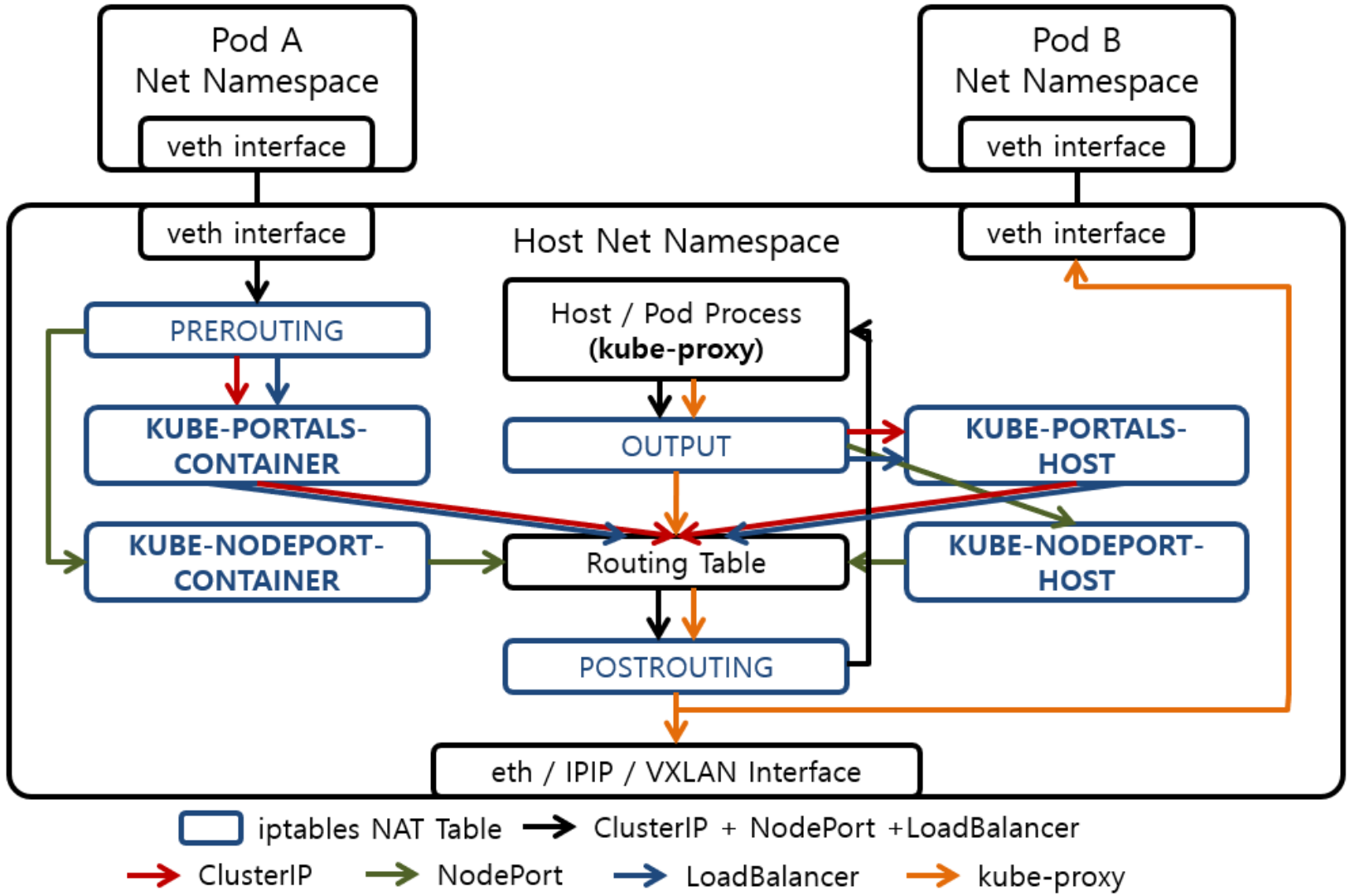

Proxy Mode
Userspace Mode
Iptables/IPVS Mode
Pod Network Area
Pod
20.111.156.6
Worker Node
Endpoint
Service
10.103.9.116
api-server
Master Node
20.111.156.6
kube-proxy
10.103.9.116
-> 20.111.156.6
10.103.9.116
iptables
Service CIDR
10.103.9.116
Pod
Worker Node
20.111.156.6
iptables
IPVS
10.103.9.116
-> 20.96.2.5
Service Network CIDR : 10.96.0.0/12
(10.96.0.0 ~ 10.111.255.255)


Service Type (ClusterIP)
192.168.59.22
eth0
Network Plugin
(cni : calico)
Overlay
IPIP
Route
NAT
calid12
calid34
eth0
20.111.156.6
Pod A
eth0
20.111.156.7
Pod B
Service
[ClusterIP]
10.103.9.116
Worker N
192.168.59.23
eth0
Outer IP: 192.168.59.22
IP : 20.111.156.7
Overlay
IPIP
IP : 20.111.156.7
Route
NAT
calid12
calid34
IP : 10.103.9.116
eth0
20.109.131.2
Pod C
eth0
20.109.131.3
Pod D
Worker Node2

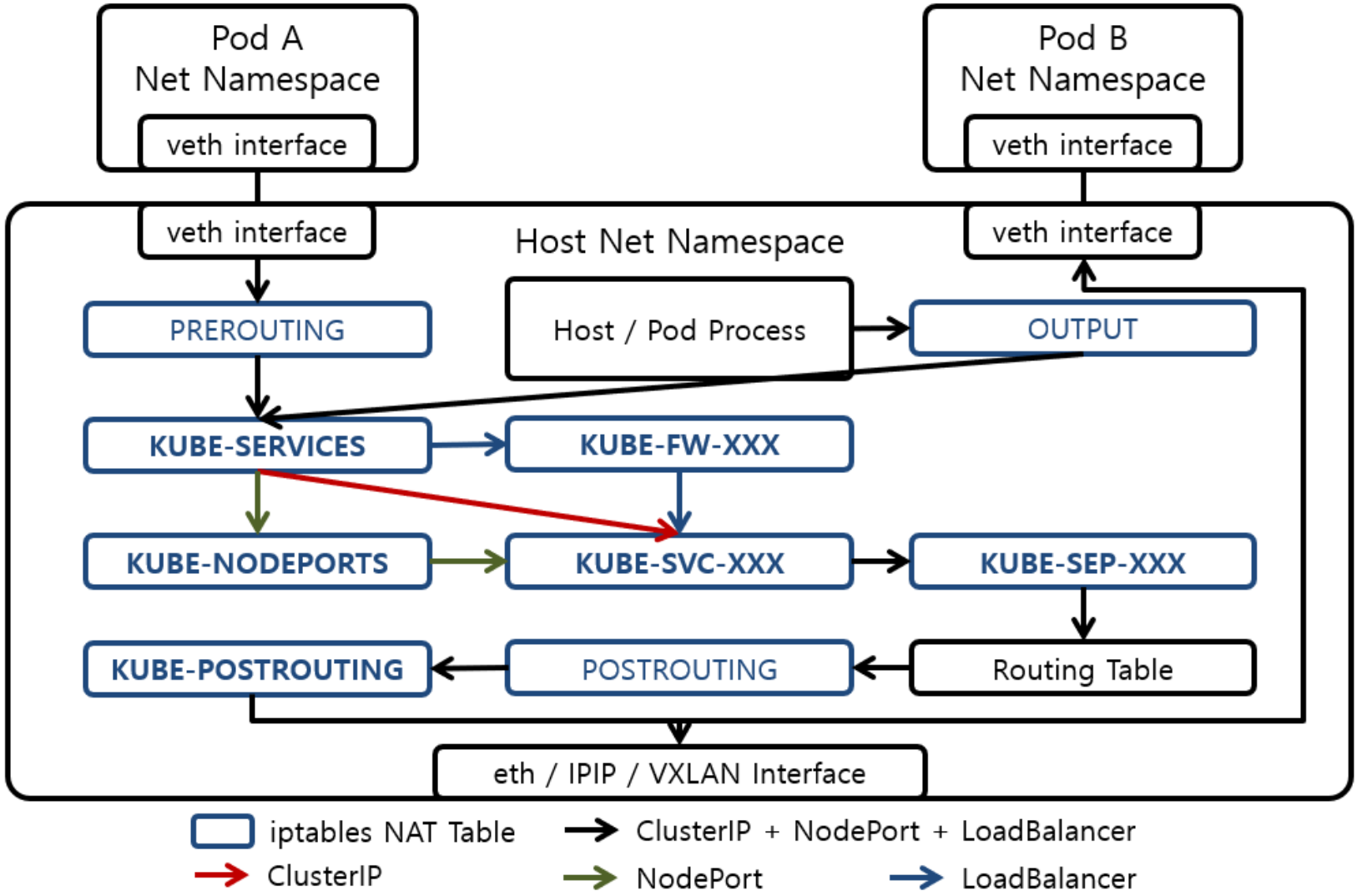

Service Type (NodePort)
IP : 192.168.59.23:38080
38080
kube-proxy
iptables
192.168.59.22
eth0
38080
192.168.59.23
eth0
Outer IP: 192.168.59.22
IP : 20.111.156.7
Network Plugin
(cni : calico)
Overlay
IPIP
IP : 20.111.156.7
Route
NAT
calid12
calid34
eth0
20.111.156.6
Pod A
eth0
20.111.156.7
Pod B
eth0
20.109.131.2
Pod C
eth0
20.109.131.3
Pod D
Service
[NodePort]
10.103.9.116
Worker N
Worker Node2

Userspace Mode

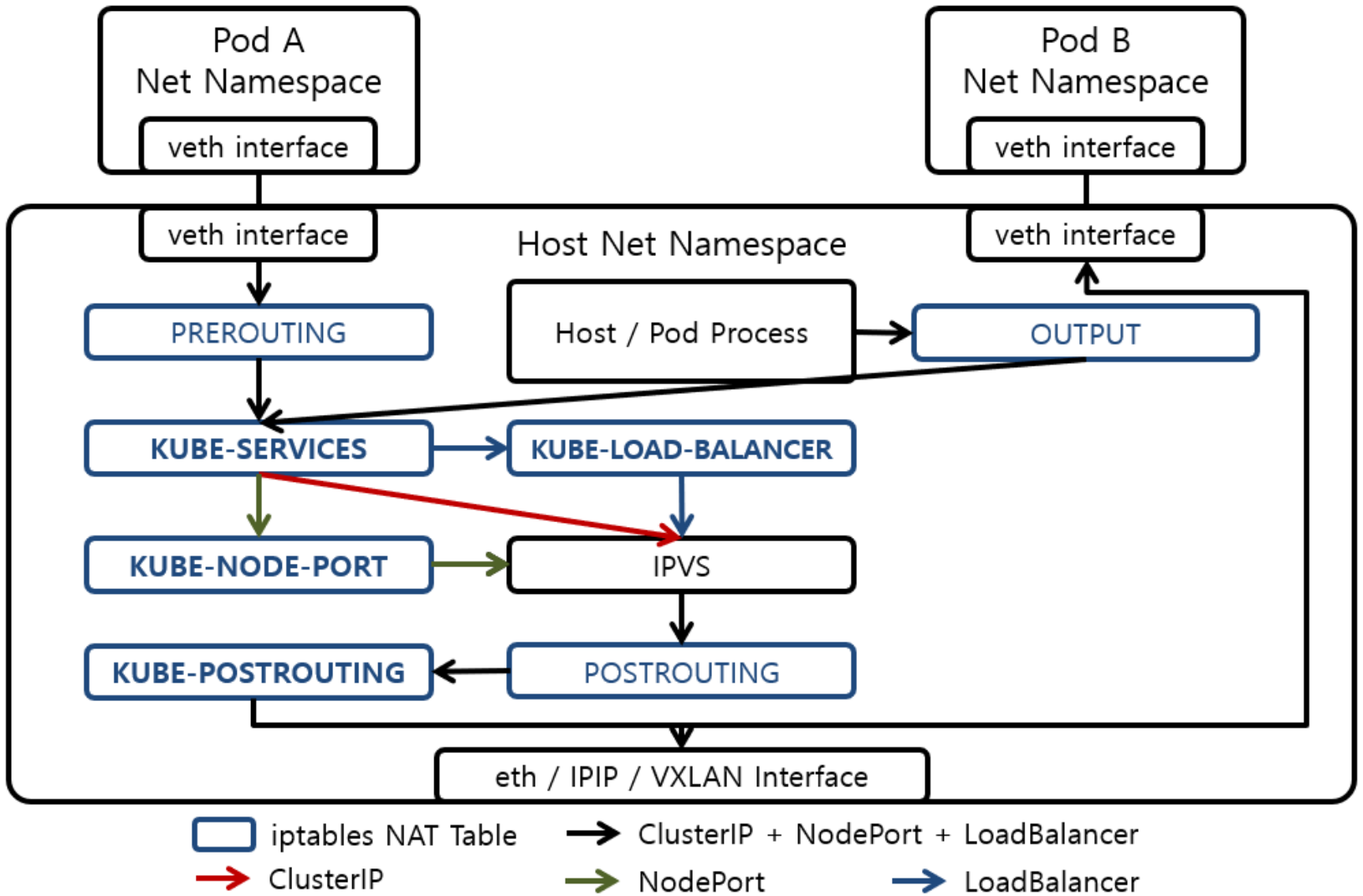
https://ssup2.github.io/theory_analysis/Kubernetes_Service_Proxy/



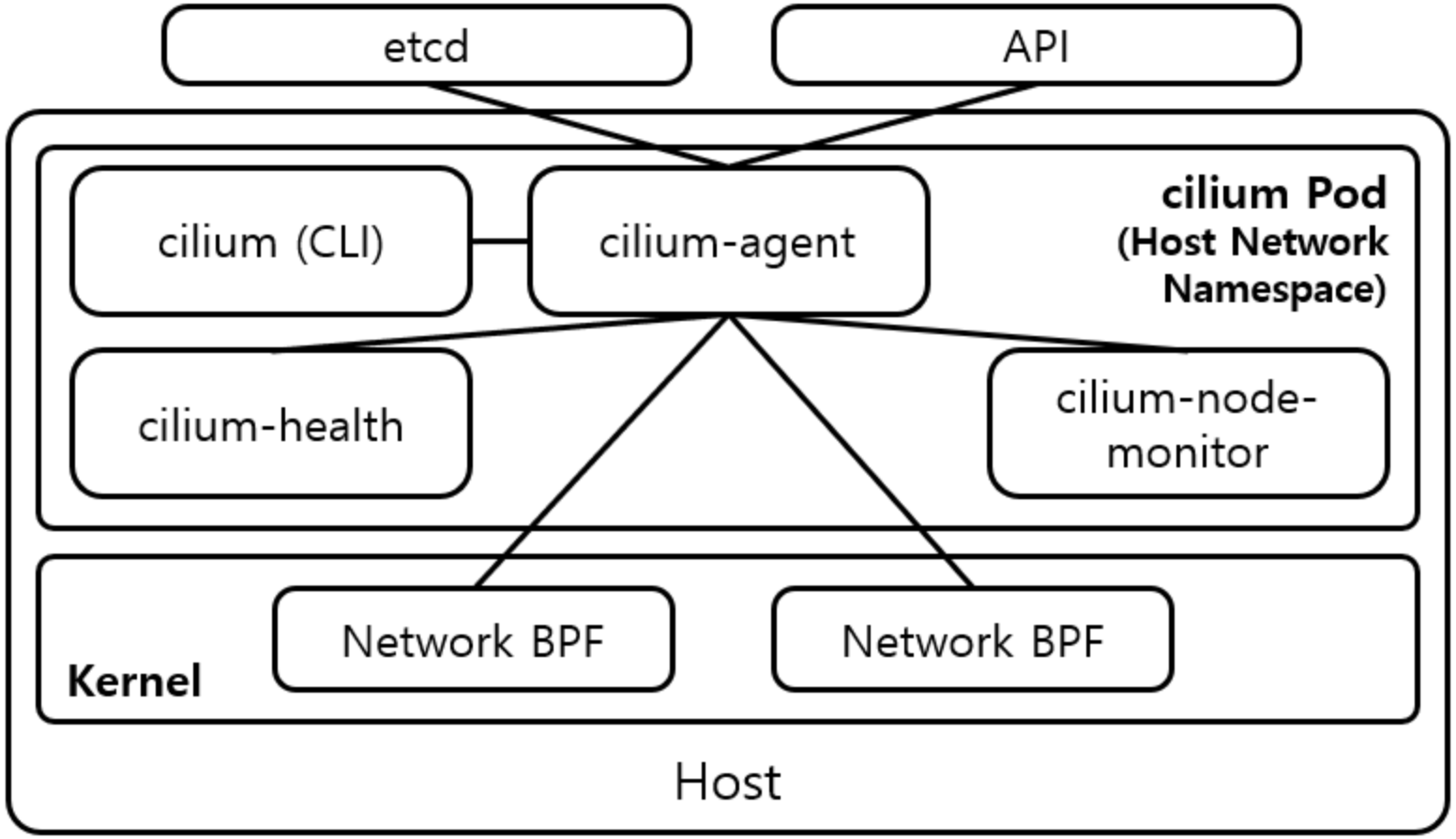
Iptables Mode



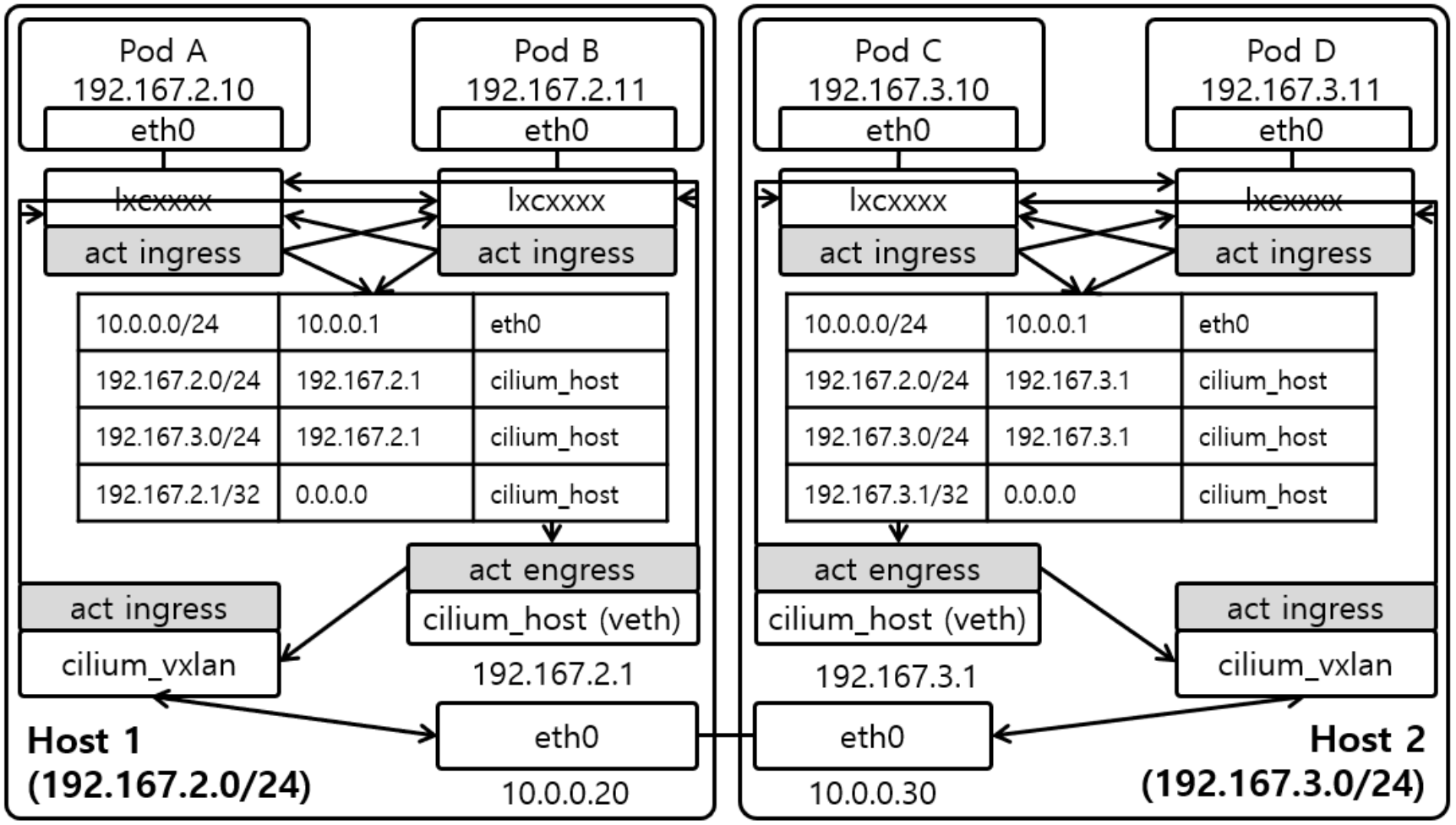
IPVS Mode


Pod A
Net Namespace
veth interface
Pod B
Net Namespace
veth interface
veth interface
Host Net Namespace
veth interface
PREROUTING
Host / Pod Process
OUTPUT
KUBE-SERVICES
KUBE-LOAD-BALANCER
KUBE-NODE-PORT
IPVS
KUBE-POSTROUTING
POSTROUTING
eth / IPIP / VXLAN Interface
iptables NAT Table
ClusterIP + NodePort + LoadBalancer
ClusterIP
NodePort
LoadBalancer

Cilium Plugin

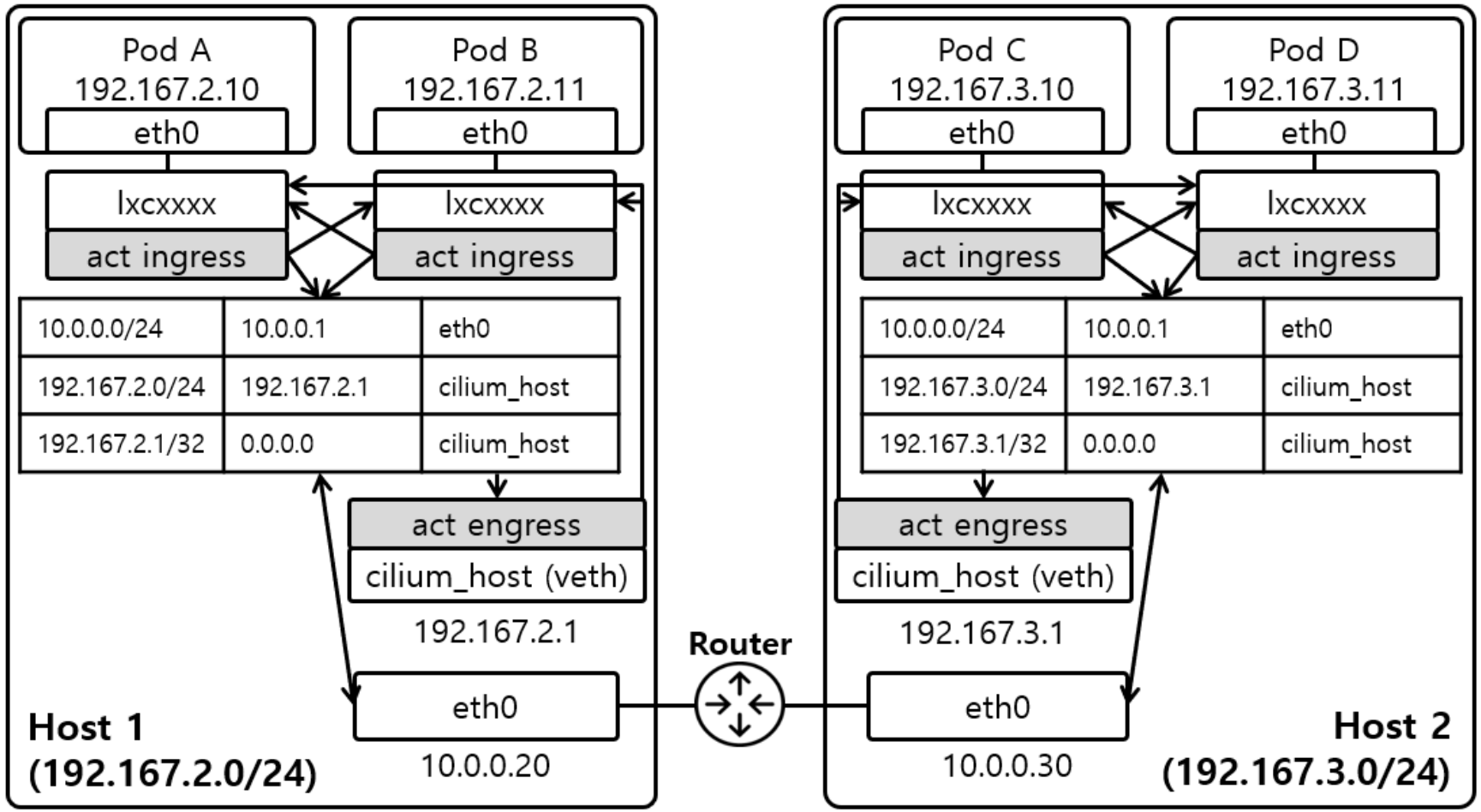


Cilium Plugin with VXLAN



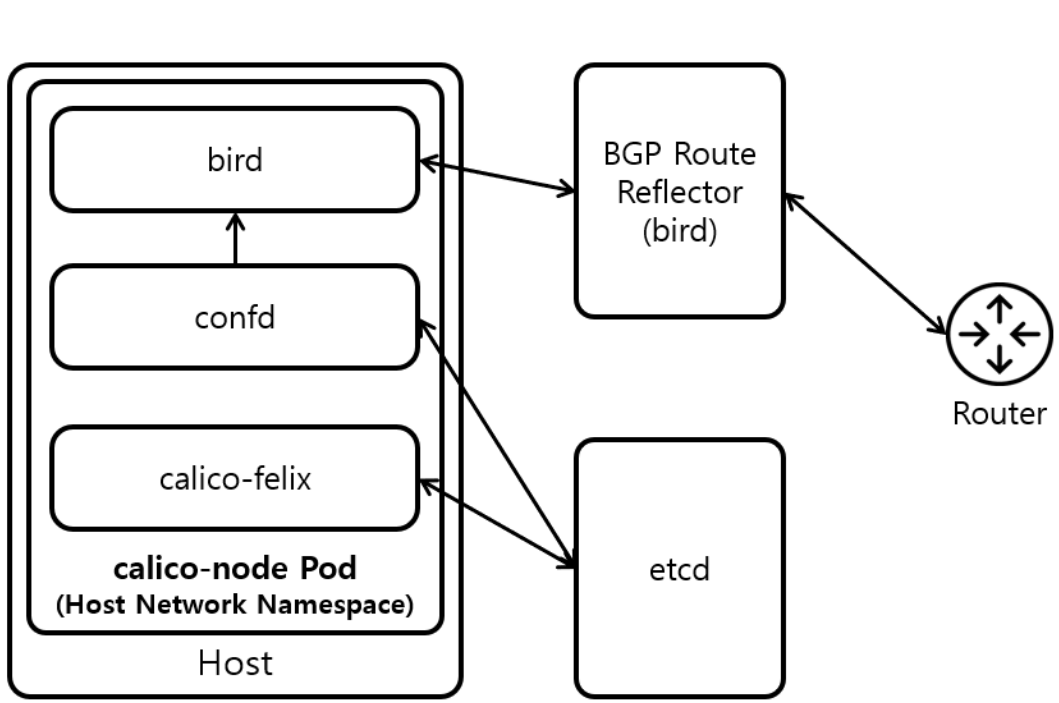
- Host Network : 10.0.0.0/24
- Container Network 192.167.0.0/24

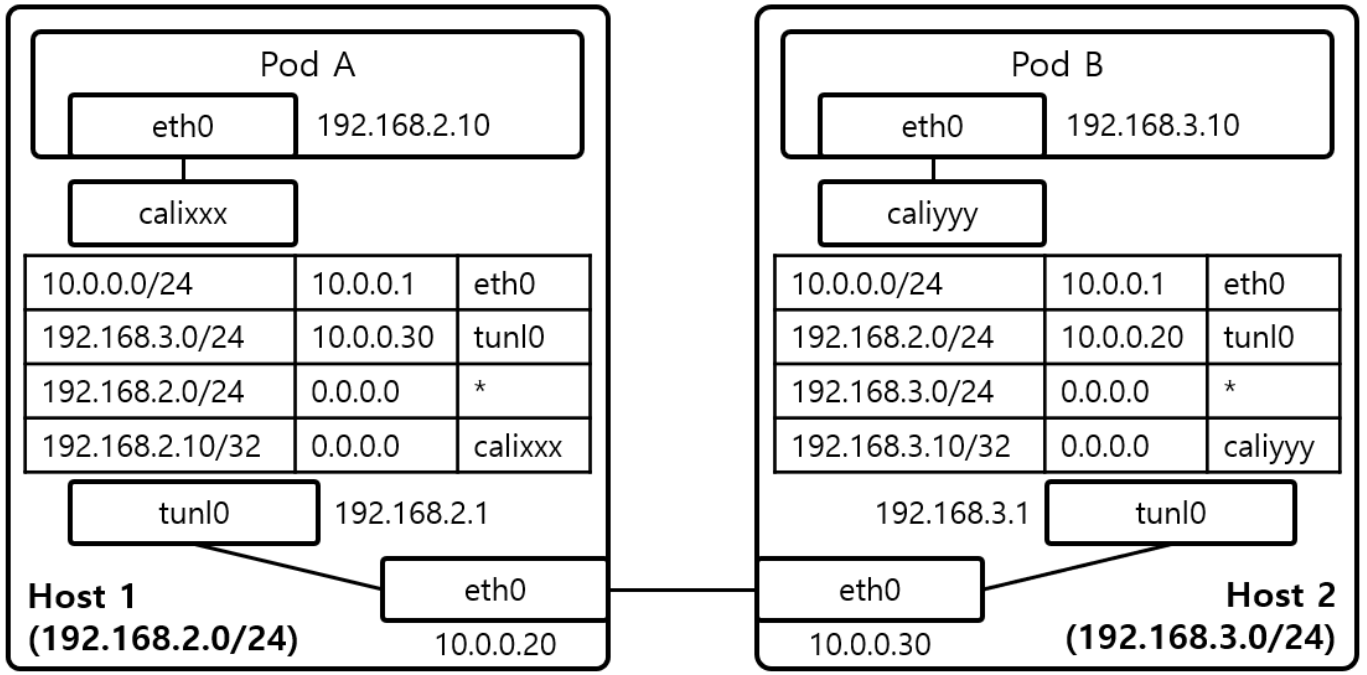
Cilium Plugin with Host L3



- Host Network : 10.0.0.0/24
- Container Network 192.167.0.0/24

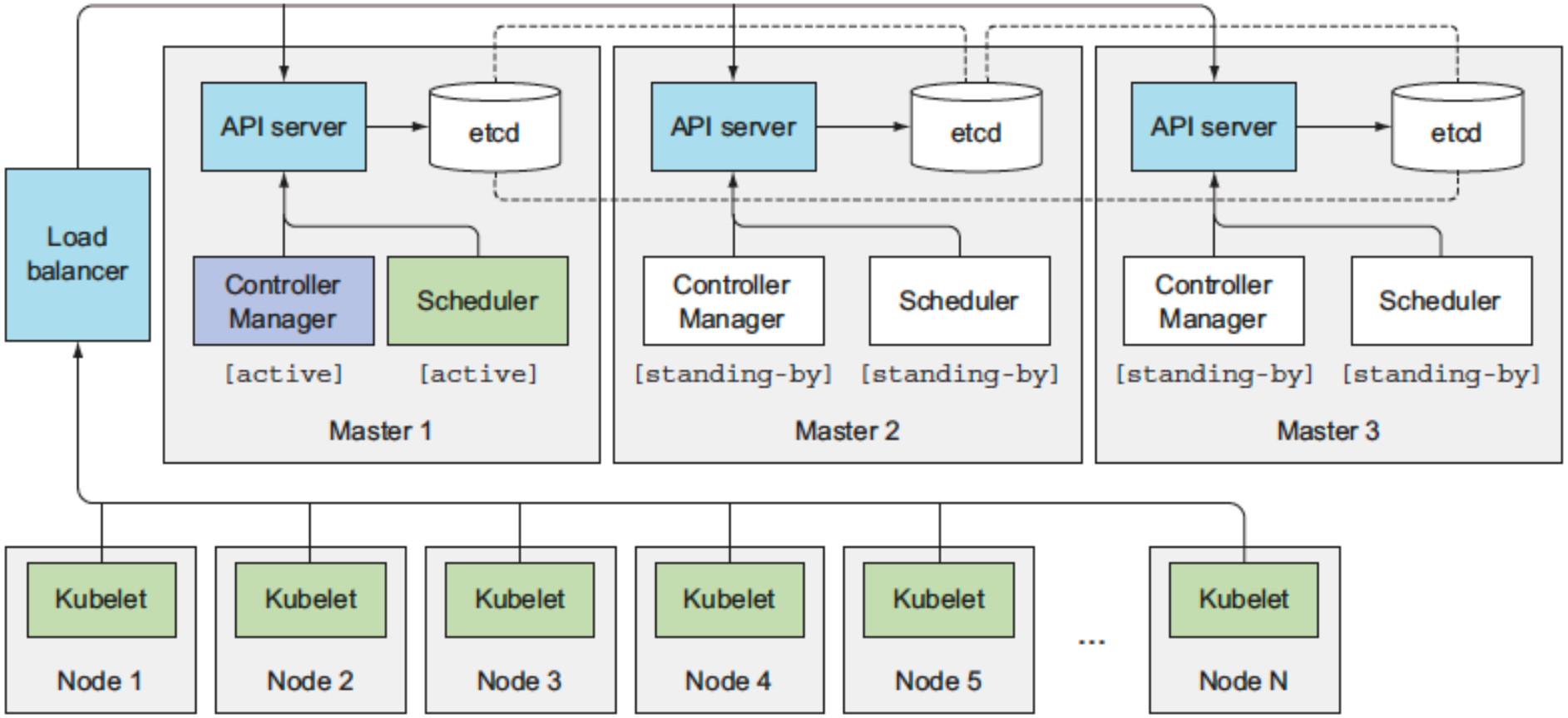
Calico Plugin



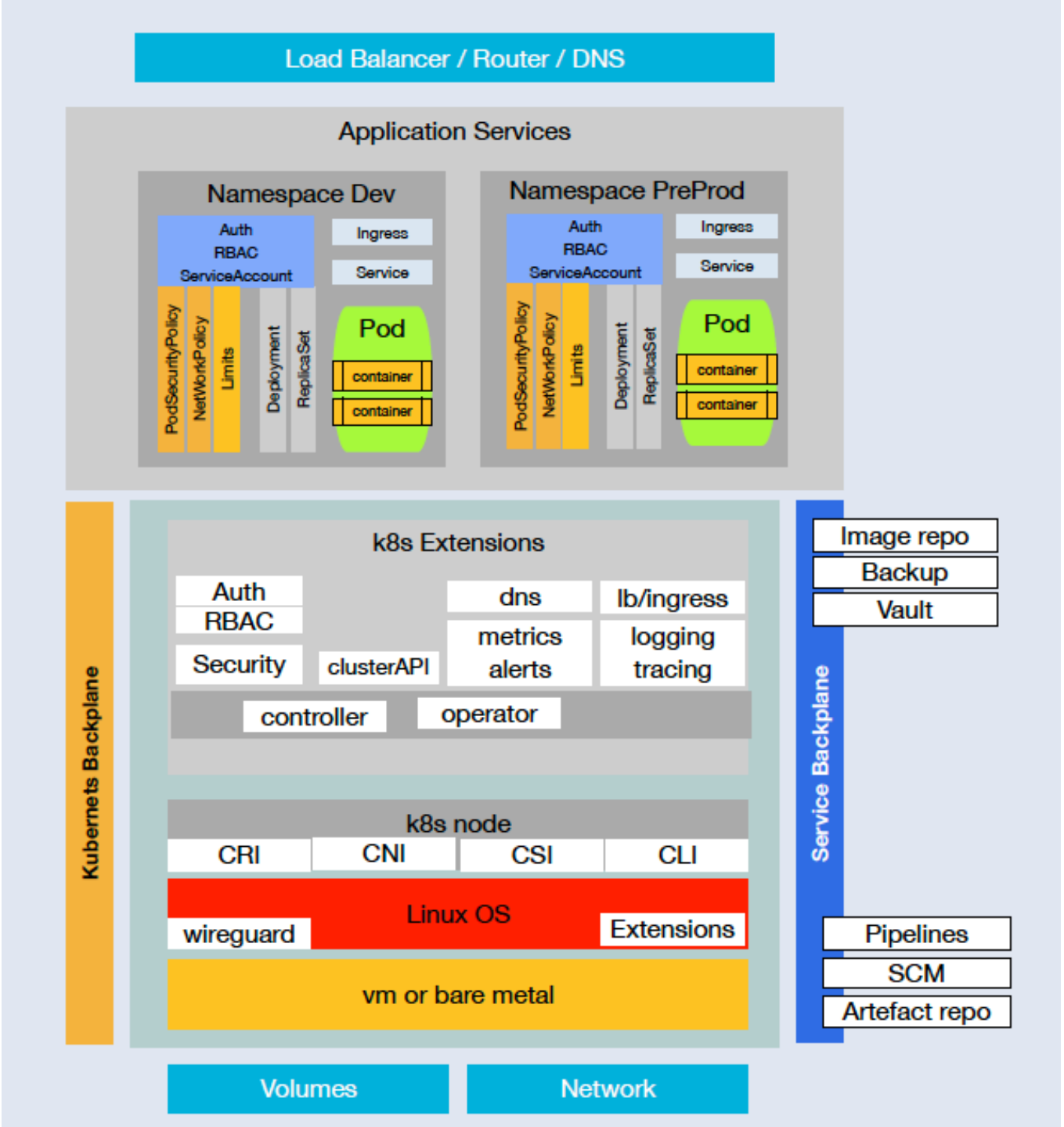


- Host Network : 10.0.0.0/24
- Pod Network 192.168.0.0/24

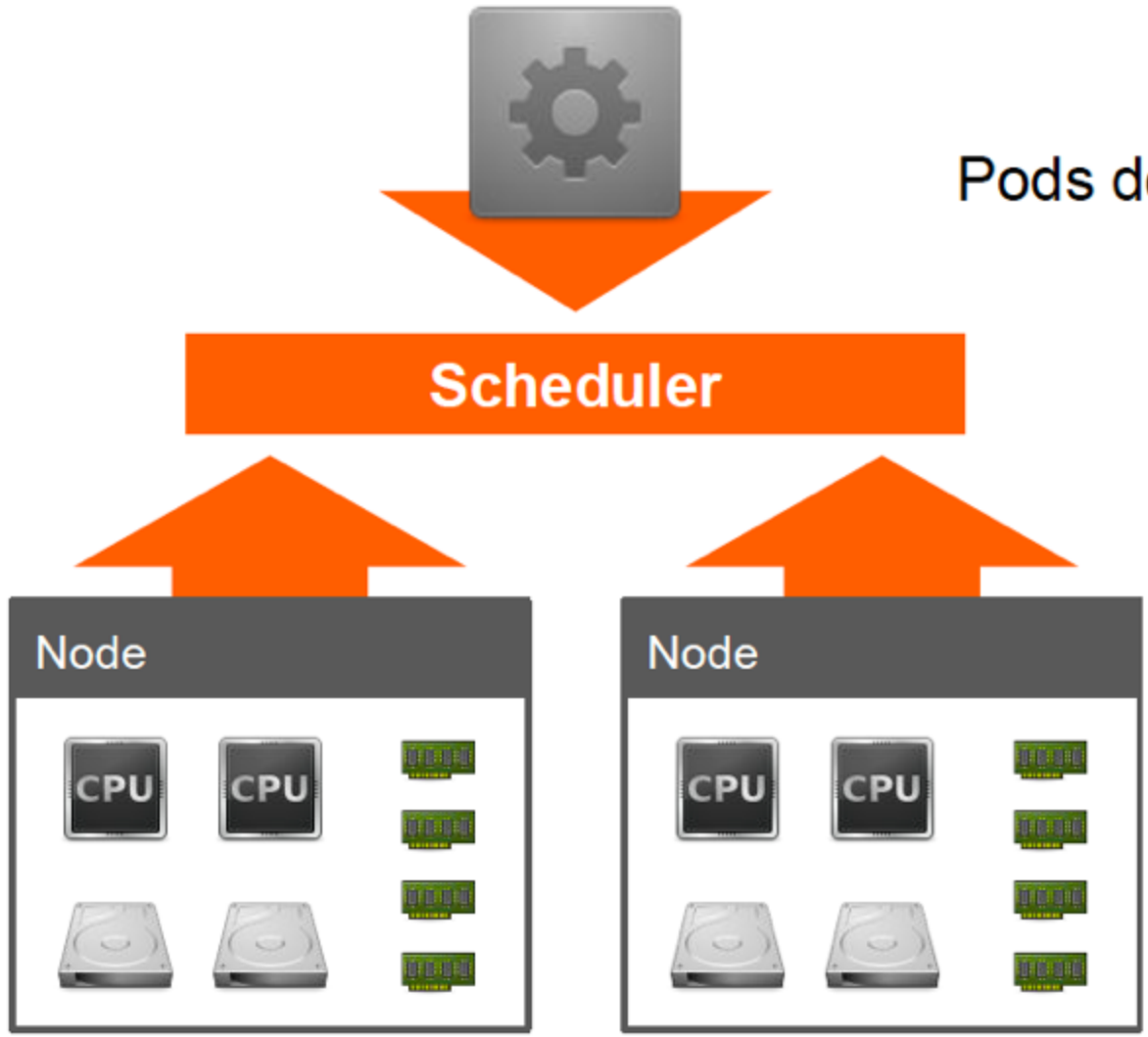
Highly Available Kubernetes Cluster



k8s Stacking



KUBERNETES: IT'S ALL ABOUT RESOURCES

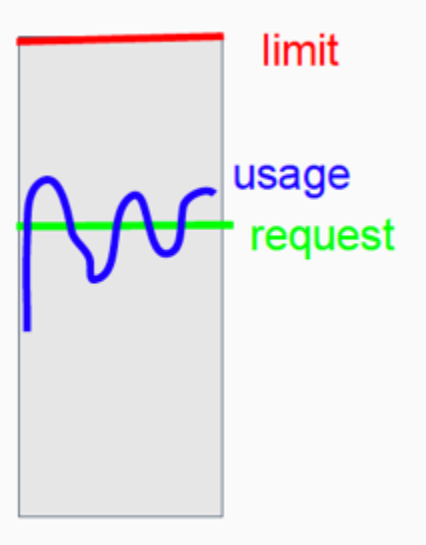


Pods demand capacity

Nodes offer capacity

zalando

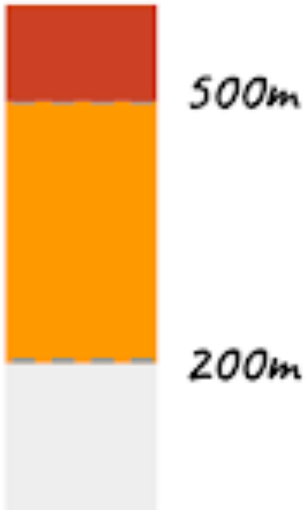
Request & Limit



CPU

Limit: where CPU throttling begins

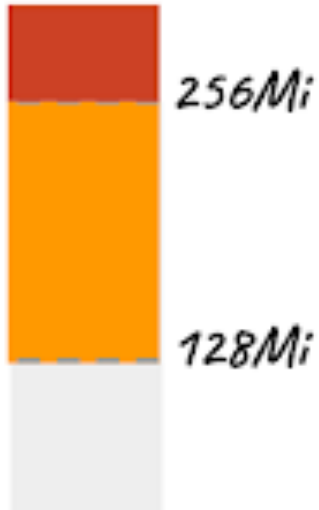
Request: amount guaranteed by k8s



Memory

Limit: level where processes are killed

Request: amount guaranteed by k8s

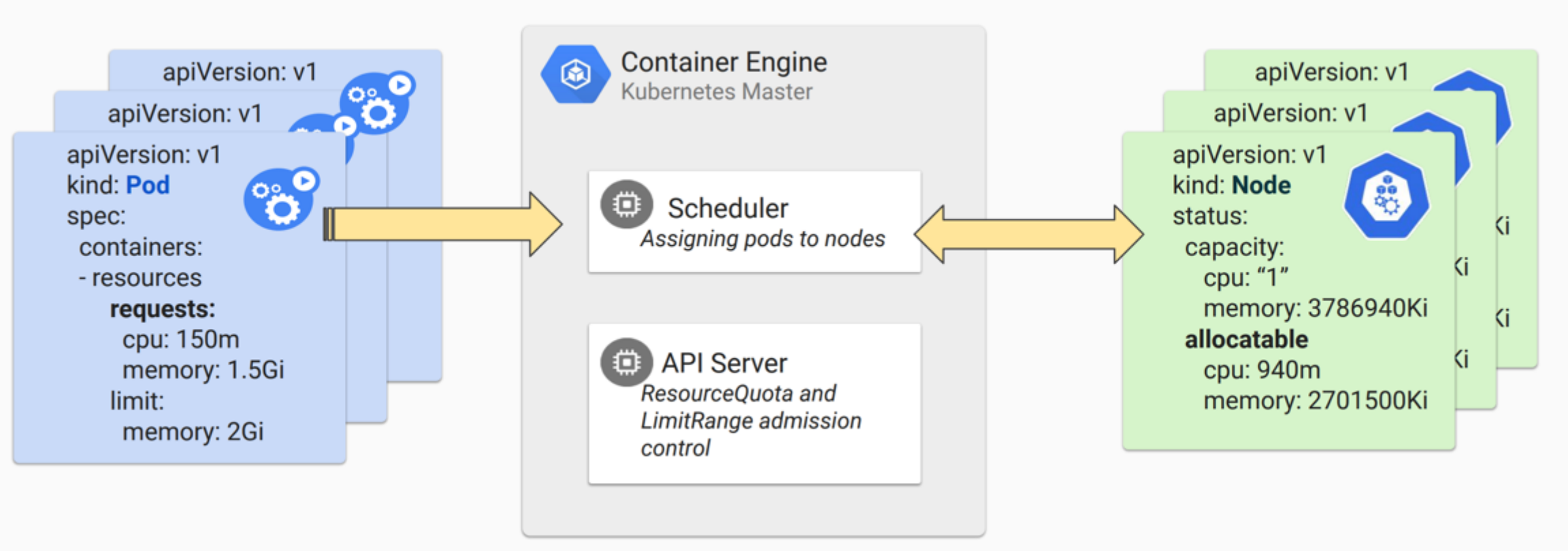


Requests, Limits by cgroup

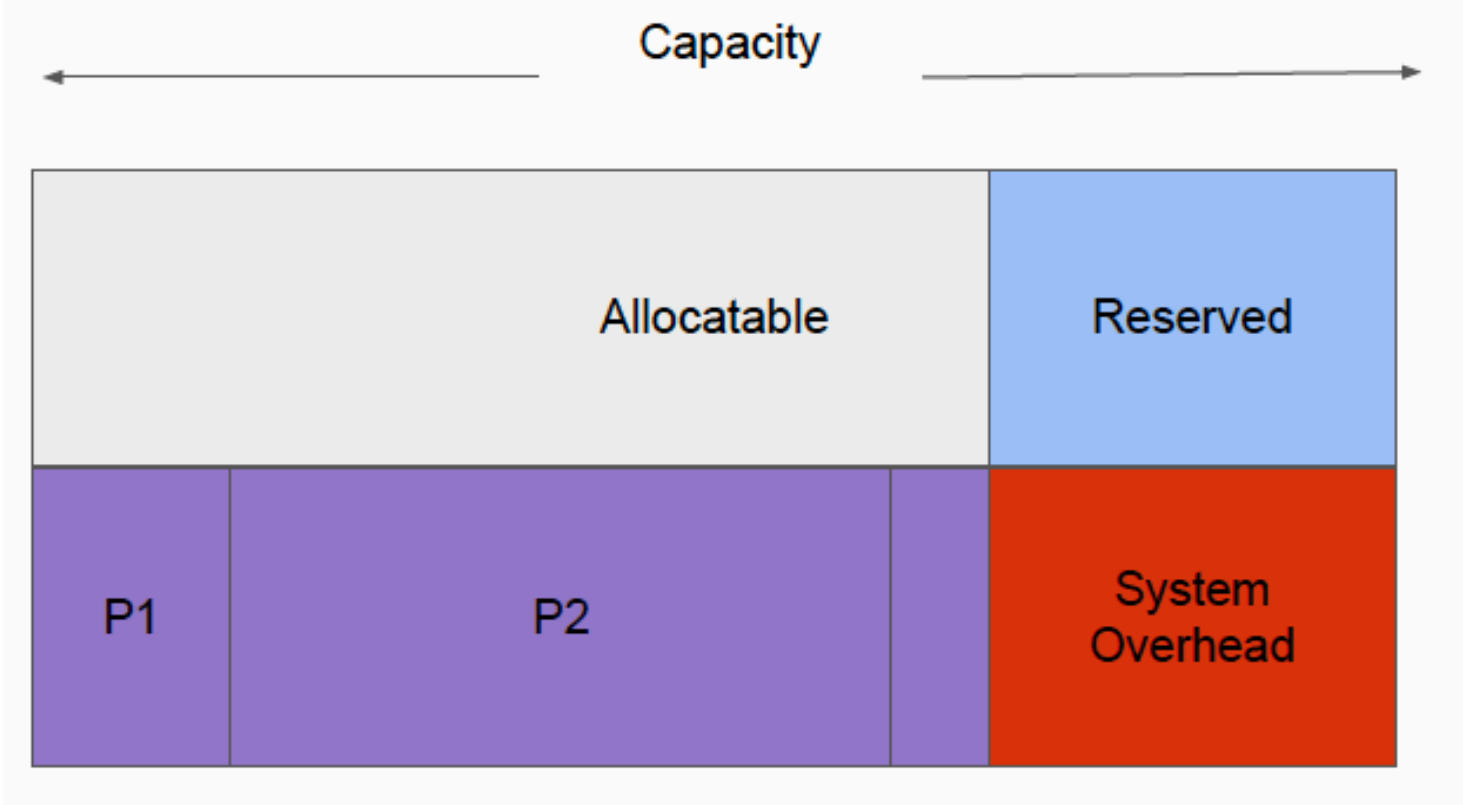
```
docker run --requests=cpu=10m/5m ..sha512()..

cat /sys/fs/cgroup/cpu/kubepods/burstable/pod5d5..0d/cpu.shares
10      // relative share of CPU time

cat /sys/fs/cgroup/cpu/kubepods/burstable/pod6e0..0d/cpu.shares
5       // relative share of CPU time

cat /sys/fs/cgroup/cpuacct/kubepods/burstable/pod5d5..0d/cpuacct.usage
    /sys/fs/cgroup/cpuacct/kubepods/burstable/pod6e0..0d/cpuacct.usage
13432815283 // total CPU time in nanoseconds
7528759332  // total CPU time in nanoseconds
```

```
docker run --cpus 1 -m 200m --rm -it busybox

cat /sys/fs/cgroup/cpu/docker/8ab25..1c/cpu.{shares,cfs_*}
1024    // cpu.shares (default value)
100000  // cpu.cfs_period_us (100ms period length)
100000  // cpu.cfs_quota_us (total CPU time in µs consumable per period)

cat /sys/fs/cgroup/memory/docker/8ab25..1c/memory.limit_in_bytes
209715200
```

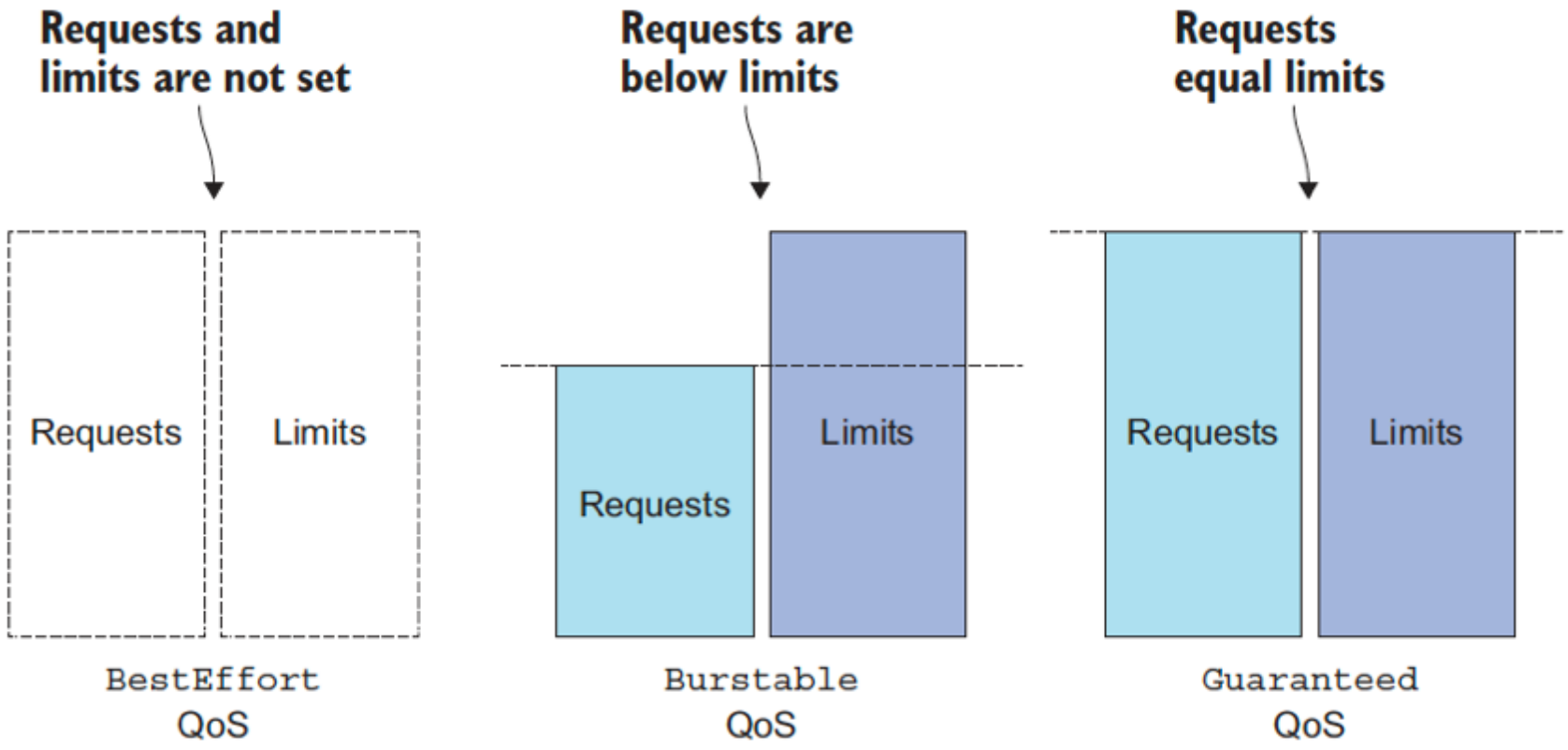
Scheduler & API Server

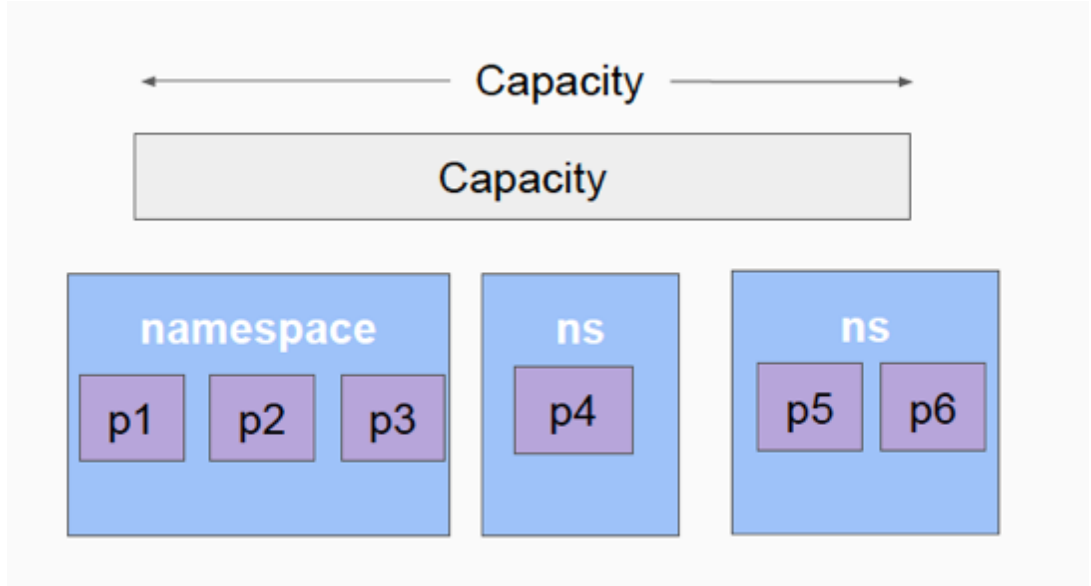


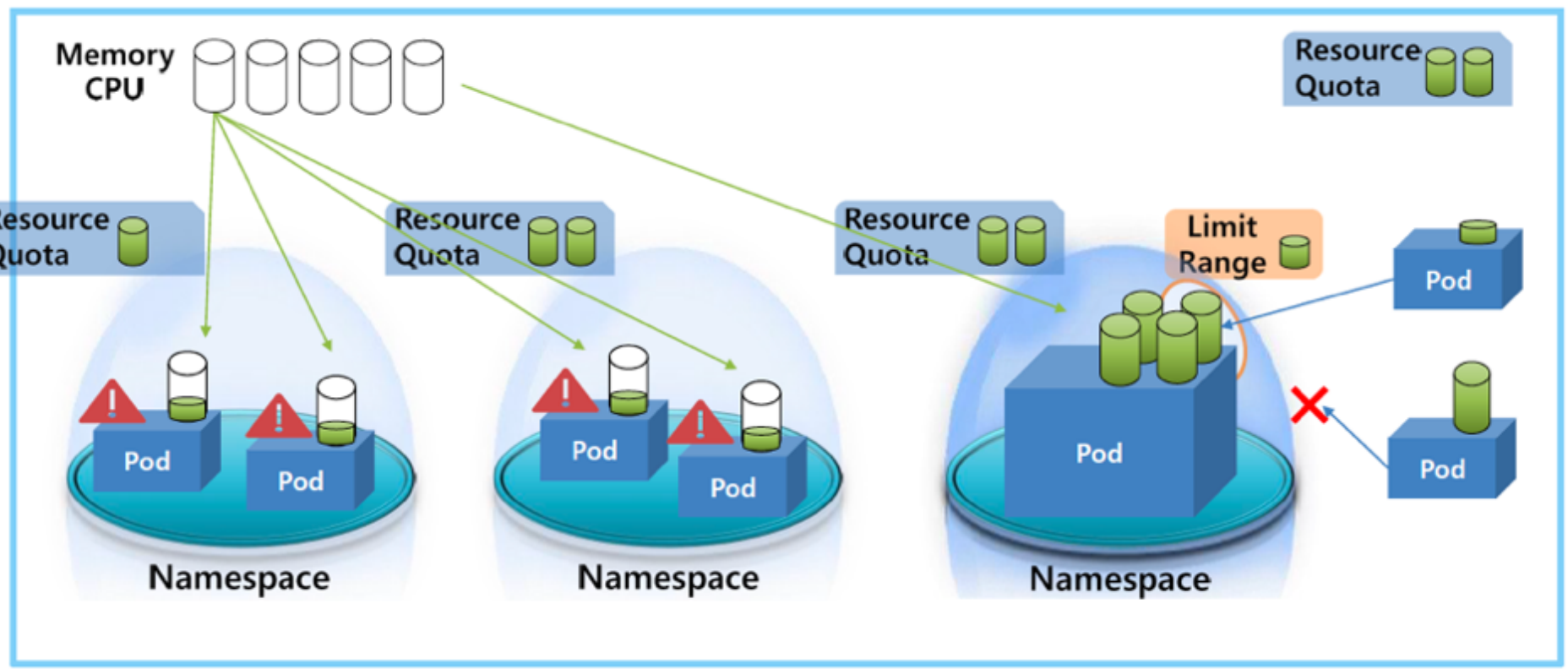
Capacity & Allocatable



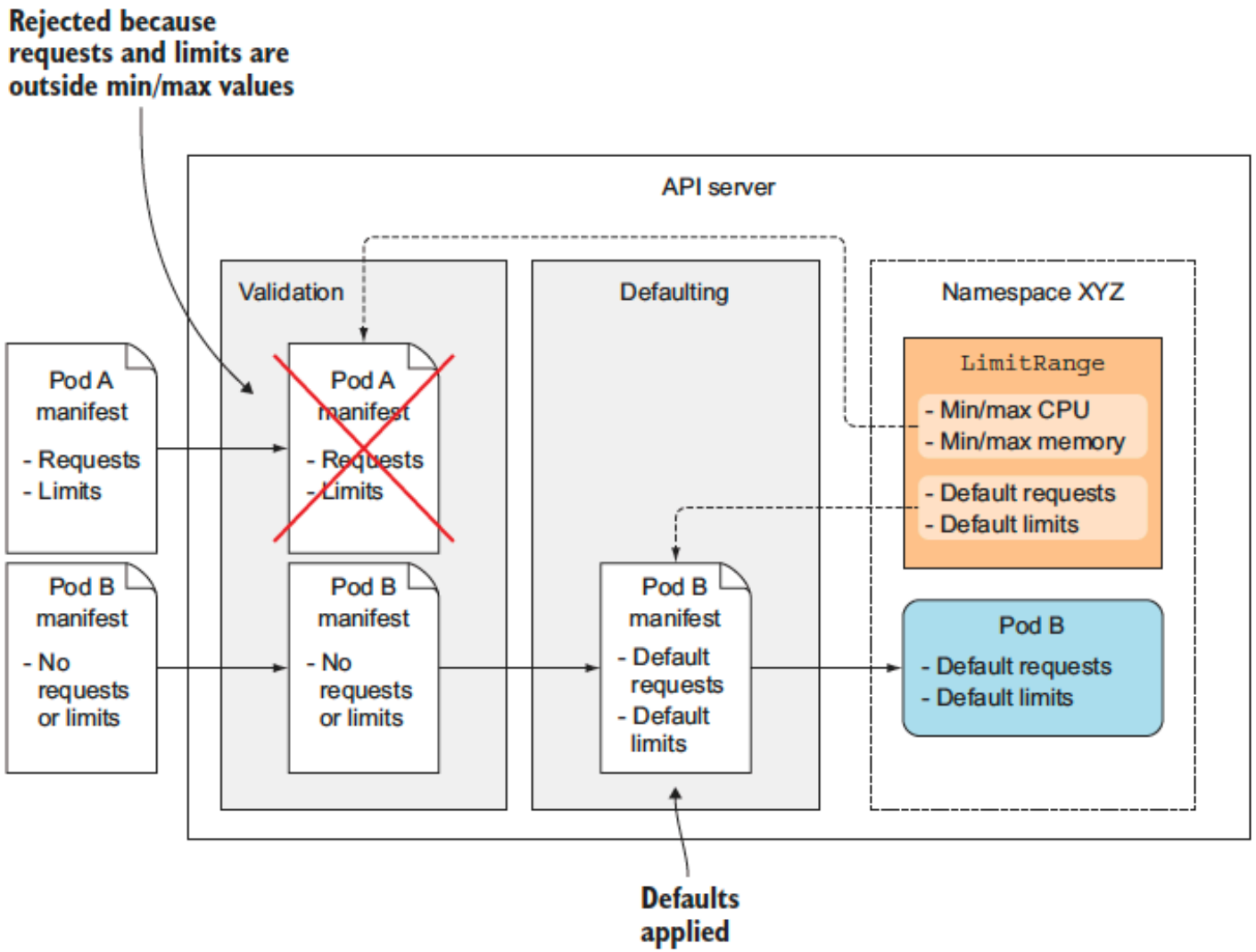
QoS: BestEffort, Burstable, Guaranteed



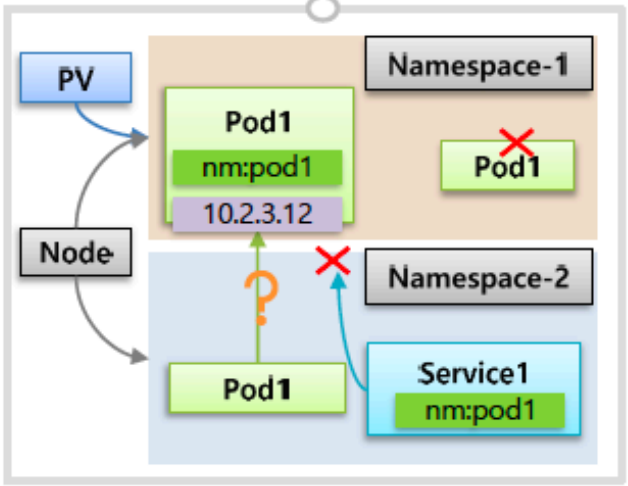

Capacity
Capacity
namespace
p1
p2
p3
ns
p4
ns
p5
p6


Memory
CPU
Resource Quota
Resource Quota
Resource Quota
Resource Quota
Limit Range
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Namespace
Namespace
Namespace

LimitRange: Admission Control



Namespace



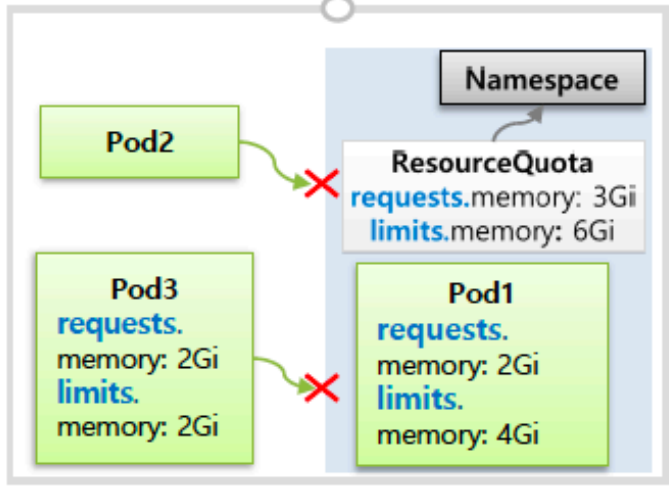

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: nm-1
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  namespace: nm-1
  labels:
    nm: pod1
spec:
  containers:
  ...
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: nm-2
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-1
  namespace: nm-2
spec:
  selector:
    nm: pod1
  ports:
  ....
```

ResourceQuota

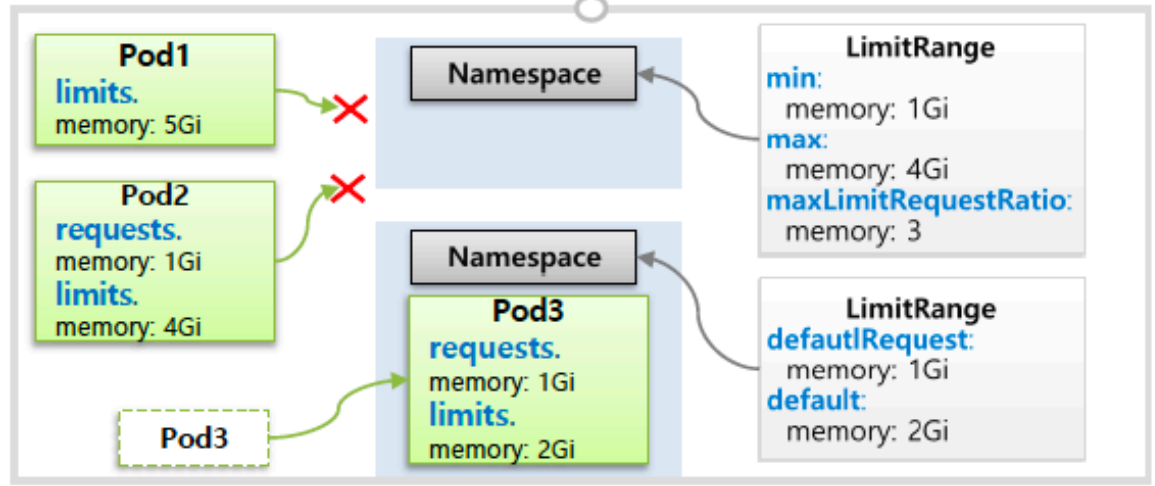



```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: rq-1
  namespace: nm-1
spec:
  hard:
    requests.memory: 3Gi
    limits.memory: 6Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-2
spec:
  containers:
  - name: container
    image:
tmkube/app
    resources:
      requests:
        memory: 2Gi
      limits:
        memory: 2Gi
```

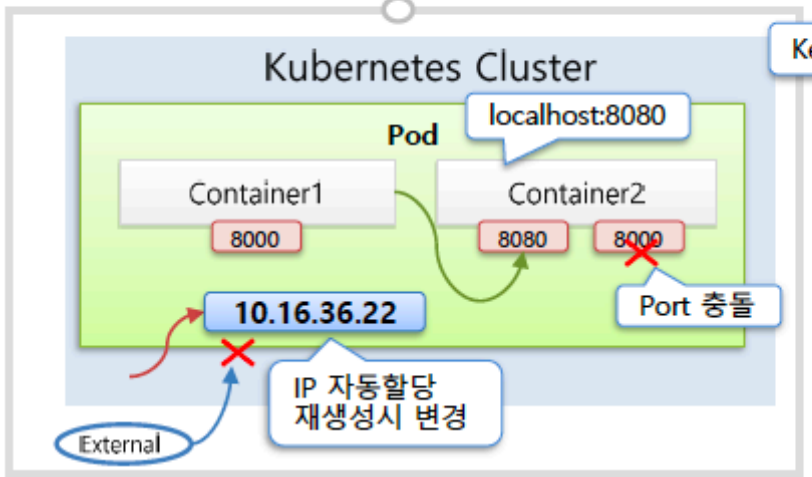
Compute Resource : cpu, memory, storage
Objects Count : Pod, Service, ConfigMap, PVC, ...

LimitRange




```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: lr-1
  namespace: nm-1
spec:
  limits:
  - type: Container
    min:
      memory: 1Gi
    max:
      memory: 4Gi
    defaultRequest:
      memory: 1Gi
    default:
      memroy: 2Gi
    maxLimitRequestRatio:
      memory: 3
```

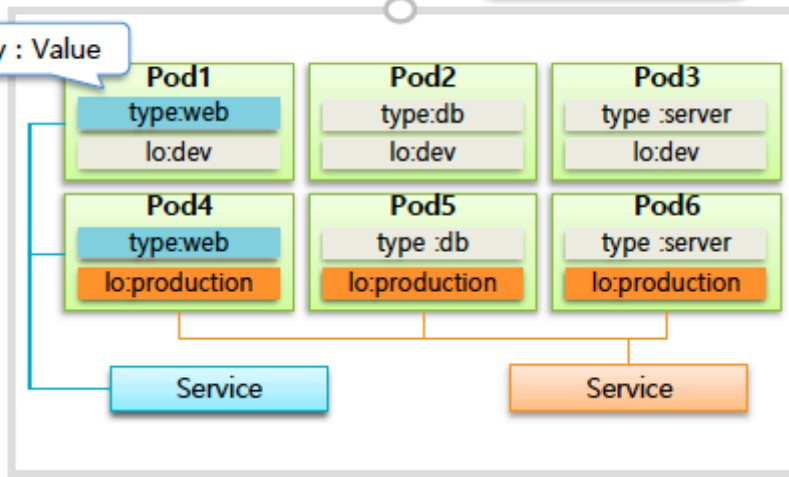
Container




```
apiVersion: v1
kind: Pod
metadata:
 name: pod-1
spec:
 containers:
 - name: container1
   image: tmkube/p8000
   ports:
   - containerPort: 8000
 - name: container2
   image: tmkube/p8080
   ports:
   - containerPort: 8080
```

Label

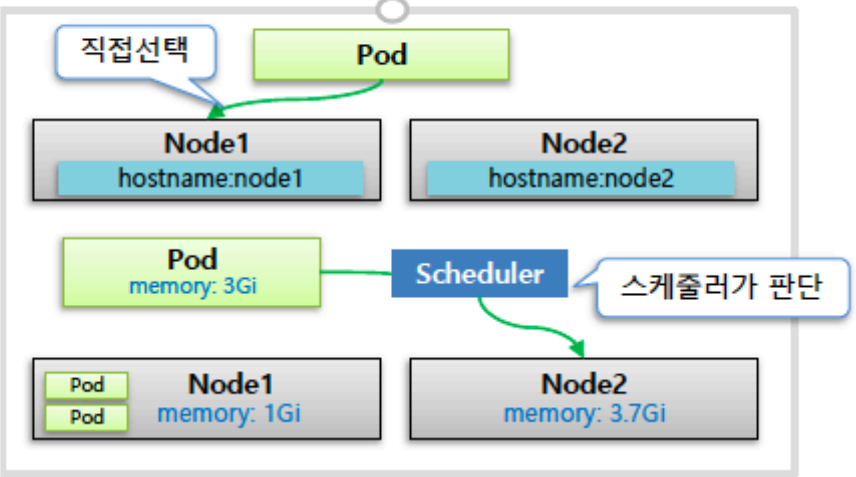
사용 목적에 따라 label 등록




```
apiVersion: v1
kind: Service
metadata:
  name: svc-1
spec:
  selector:
    type: web
  ports:
    - port: 8080
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-2
  labels:
    type: web
    lo: dev
spec:
  containers:
  - name: container
    image: tmkube/init
```

Node Schedule

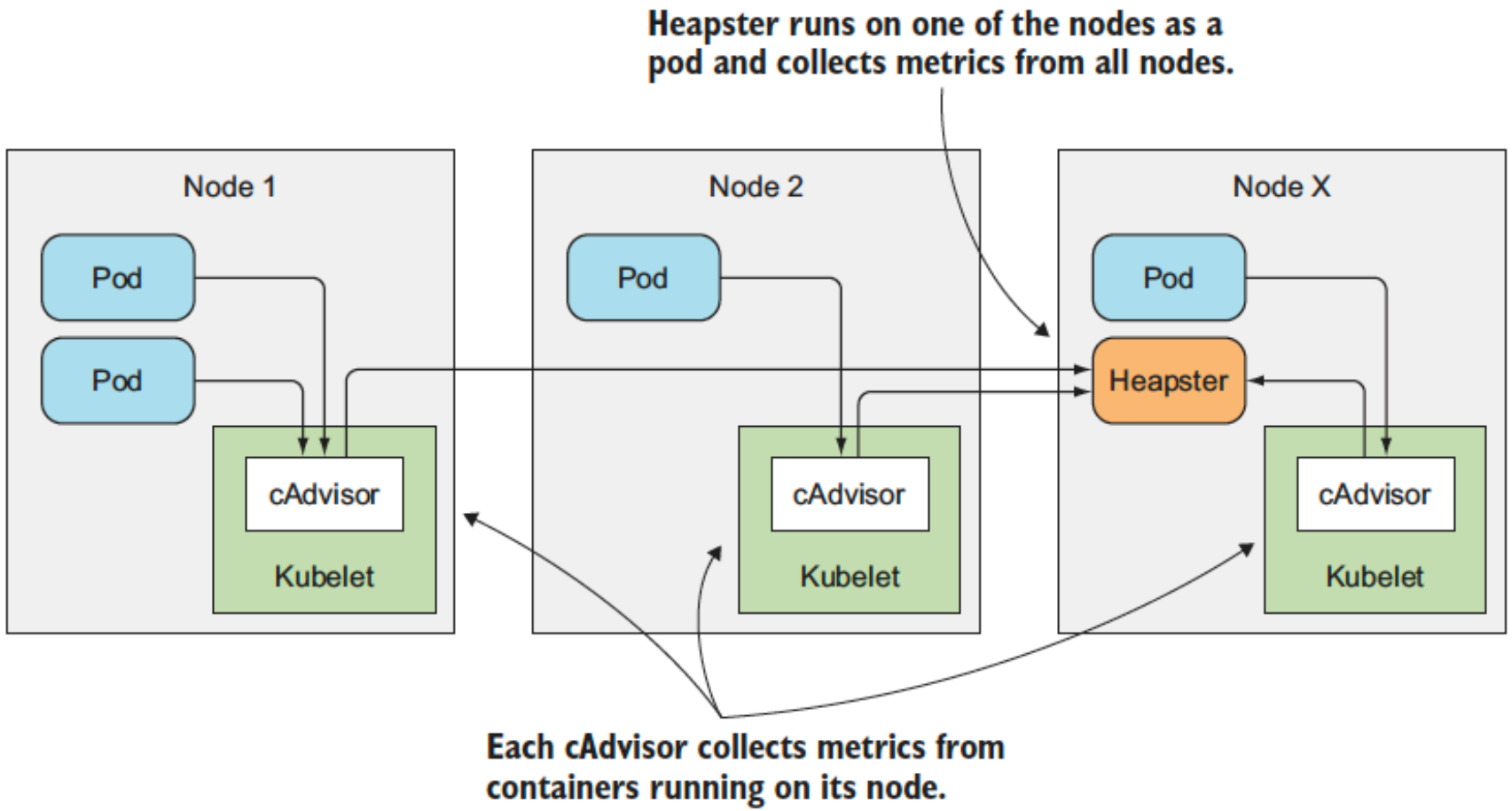



```
apiVersion: v1
kind: Pod
metadata:
 name: pod-3
spec:
 nodeSelector:
  hostname: node1
 containers:
 - name: container
   image: tmkube/init
```

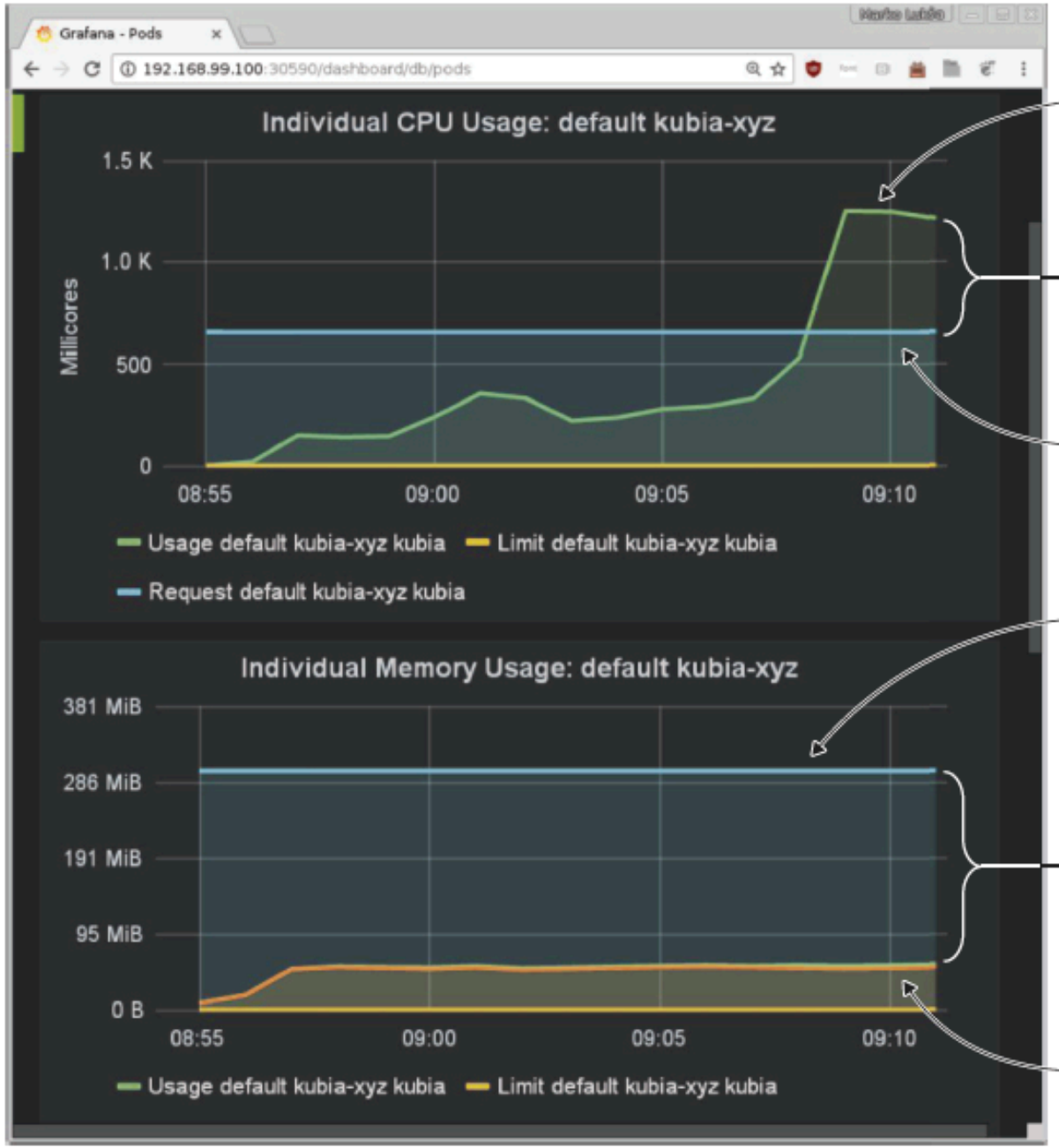
```
apiVersion: v1
kind: Pod
metadata:
  name: pod-4
spec:
  containers:
  - name: container
    image: tmkube/init
    resources:
      requests:
        memory: 2Gi
      limits:
        memory: 3Gi
```

Memory : 초과시 Pod 종료 시킴
Cpu : 초과시 request로 낮춤
Over시 종료되지 않음

cAdvisor & Heapster



Grafana



CPU usage

Actual CPU usage is higher than what was requested. The application's CPU time will be throttled when other apps demand more CPU. You should increase the CPU request.

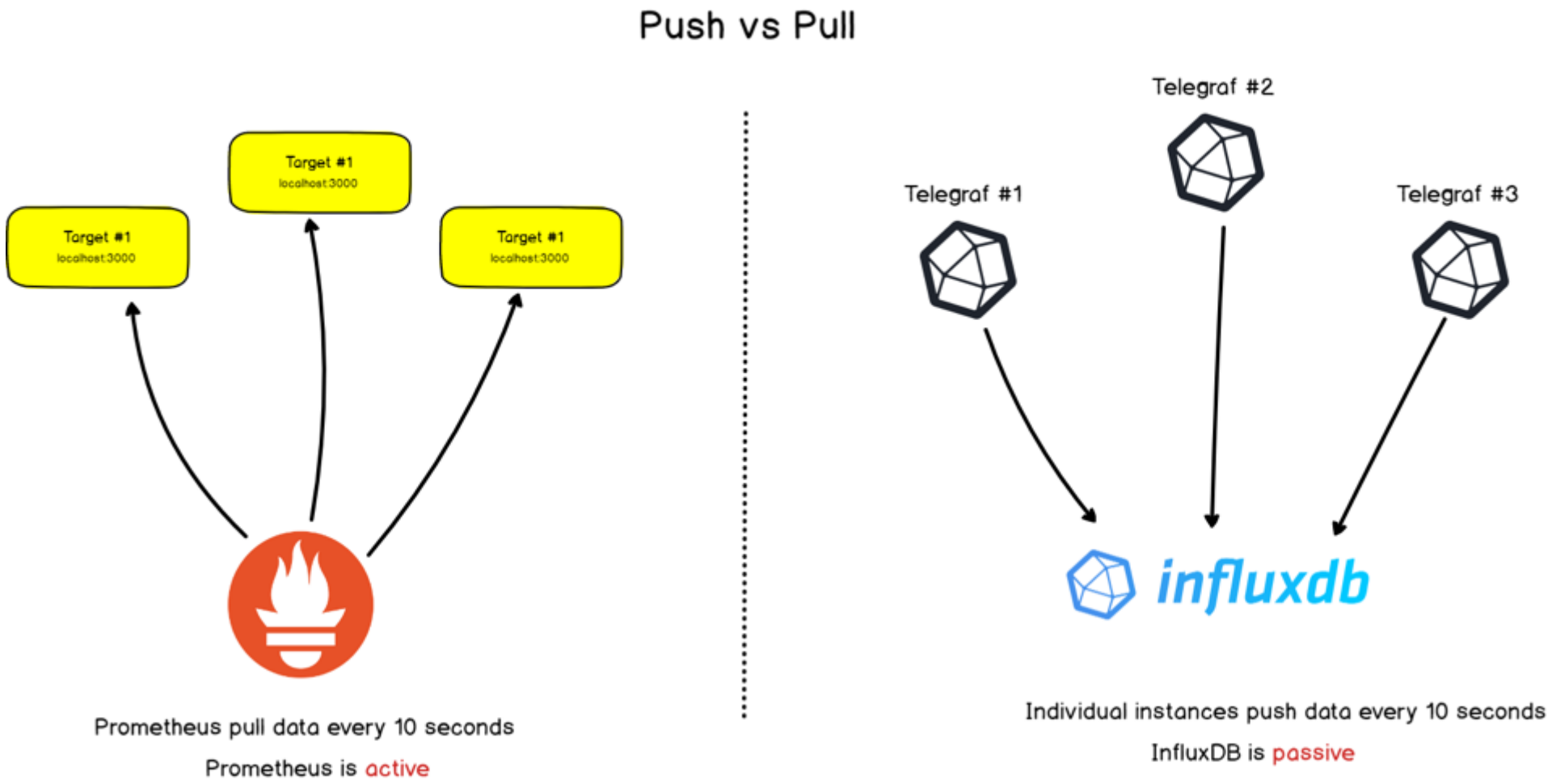
CPU request

Memory request

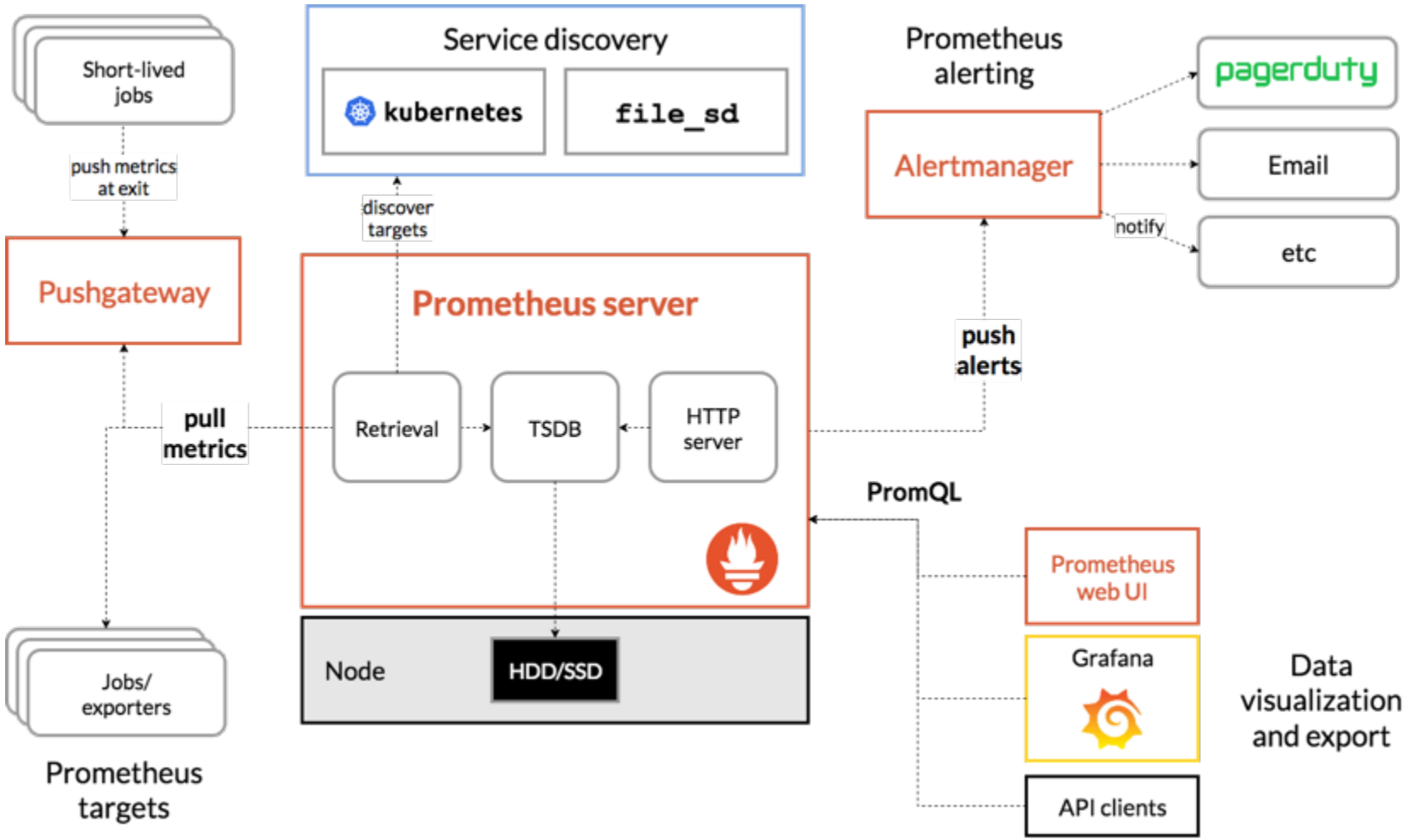
Actual memory usage is well below requested memory. You've reserved too much memory for this app. You're wasting memory, because it won't ever be used by this app and also can't be used by other apps. You should decrease the memory request.

Memory usage

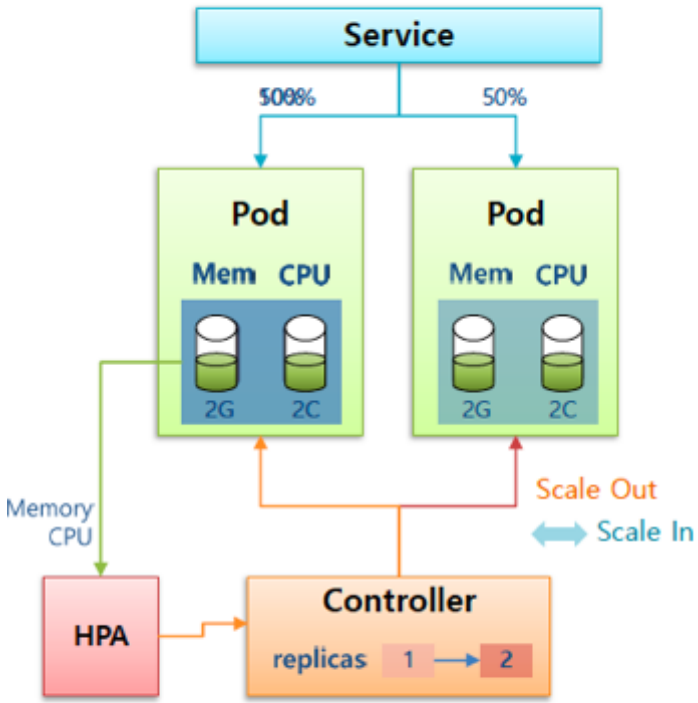
Prometheus vs influxdb



Prometheus

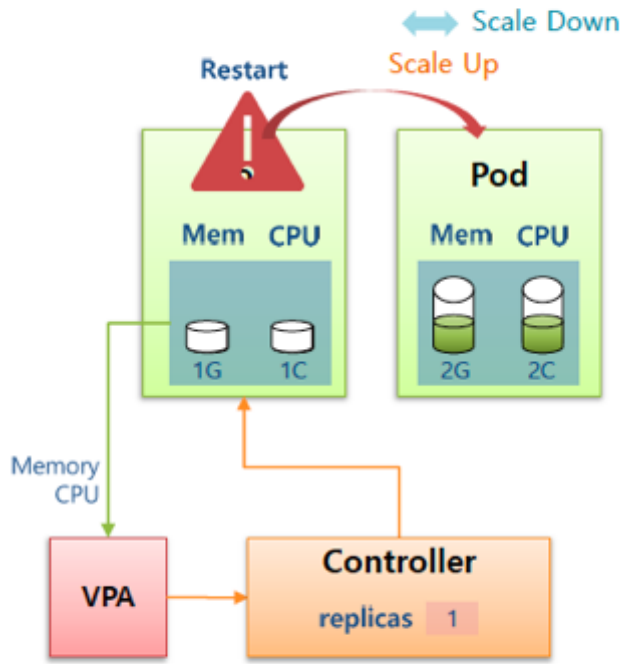

Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

HPA (Horizontal Pod Autoscaler)



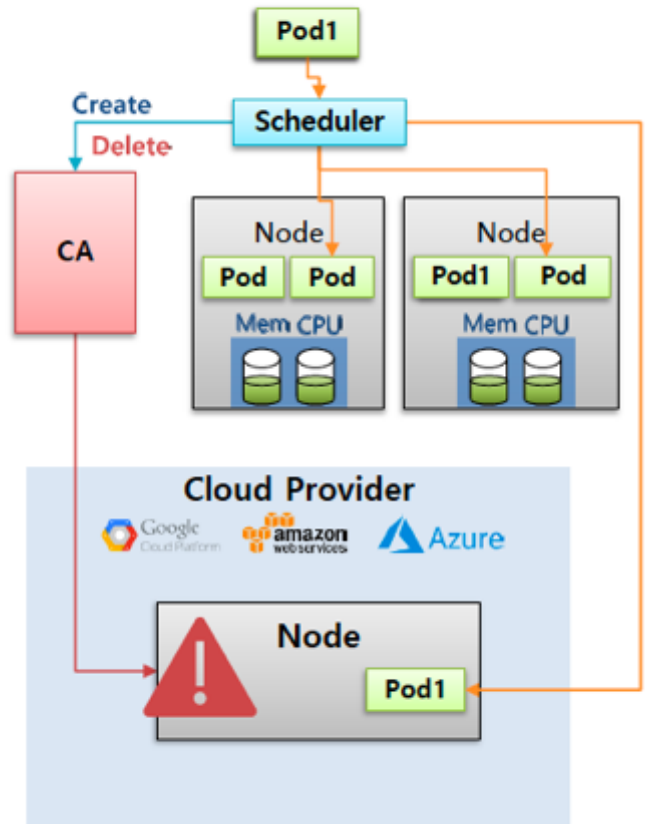
- 기동이 빠르게 되는 App
- Stateless App

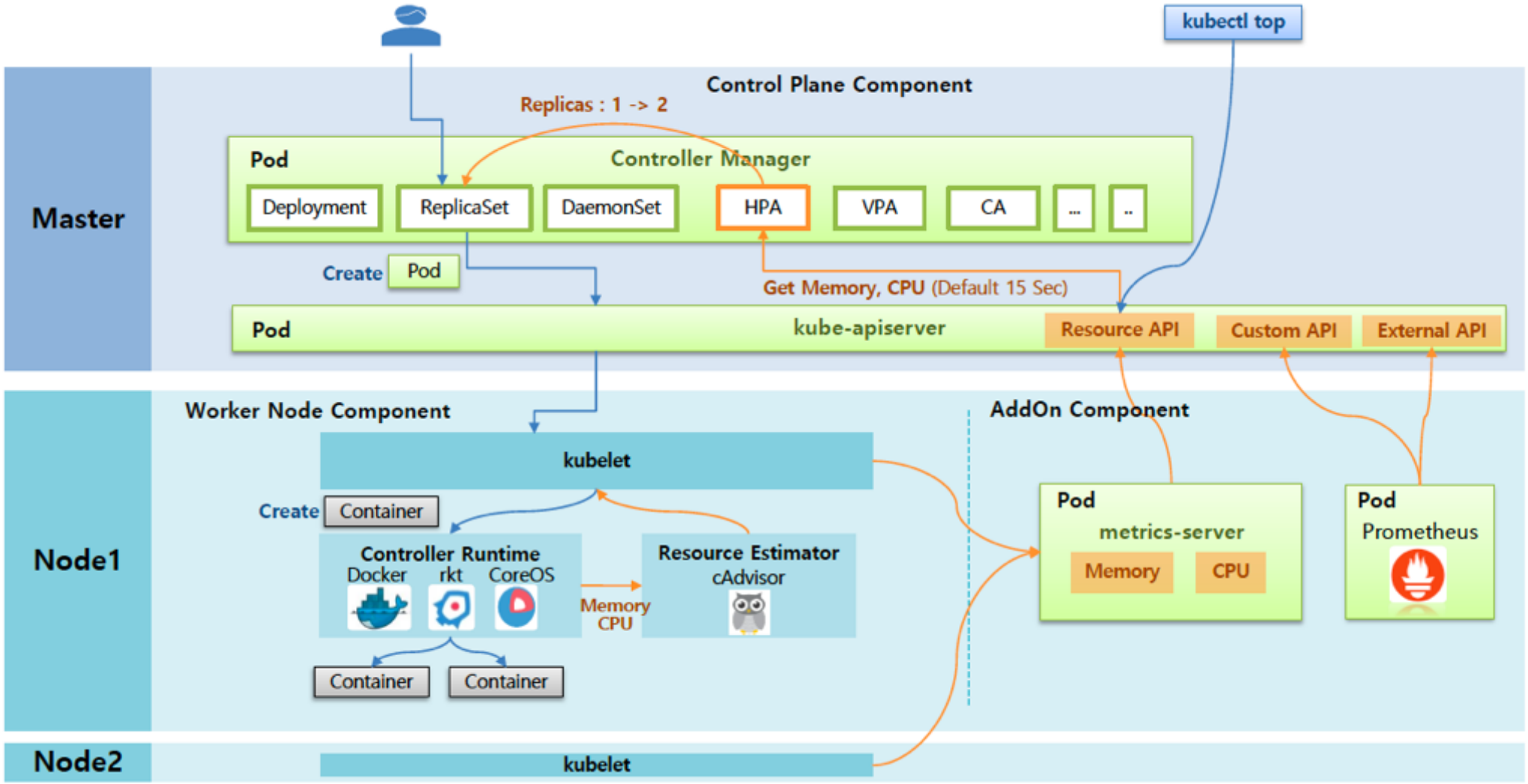
VPA (Vertical Pod Autoscaler)

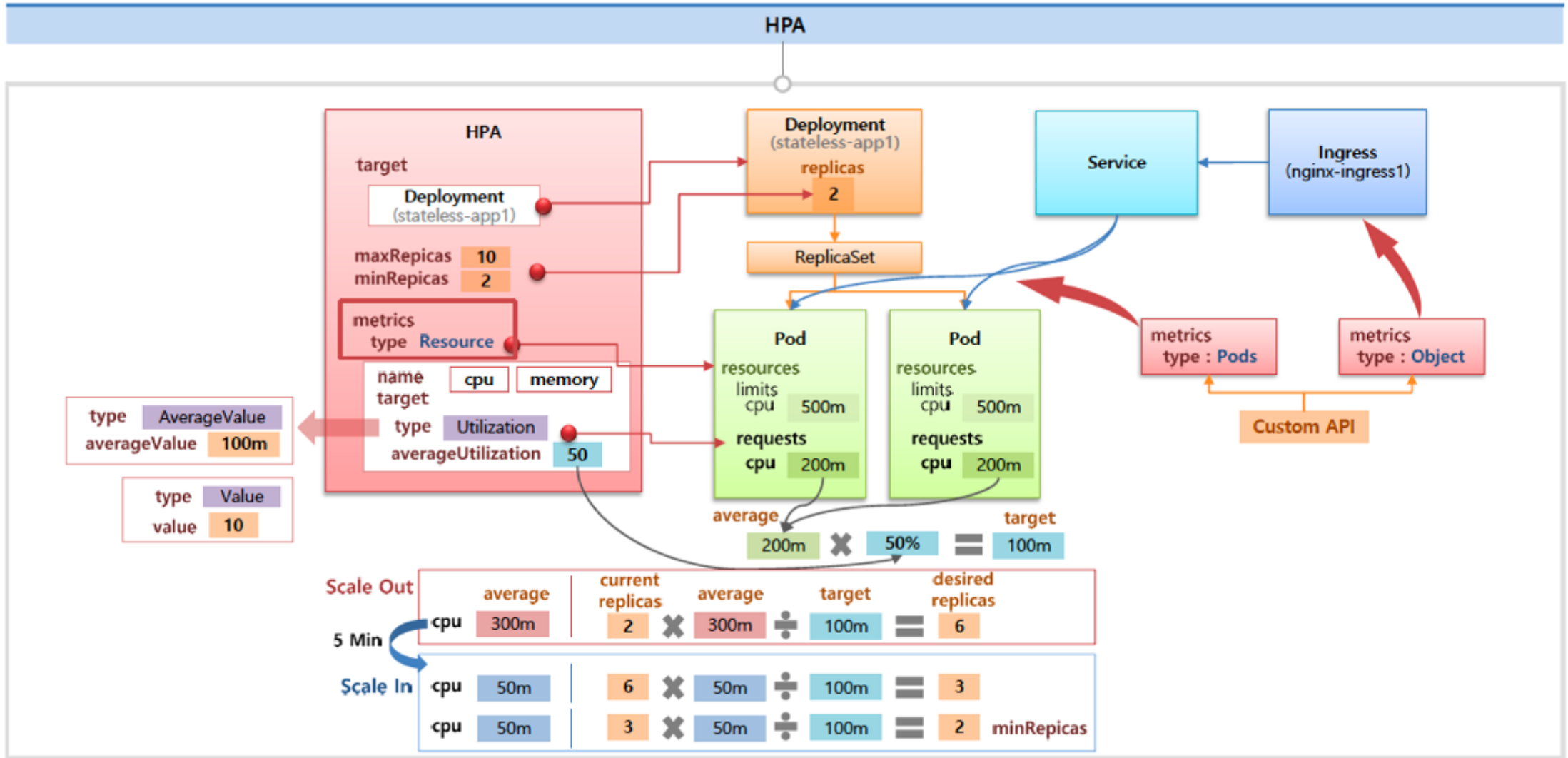


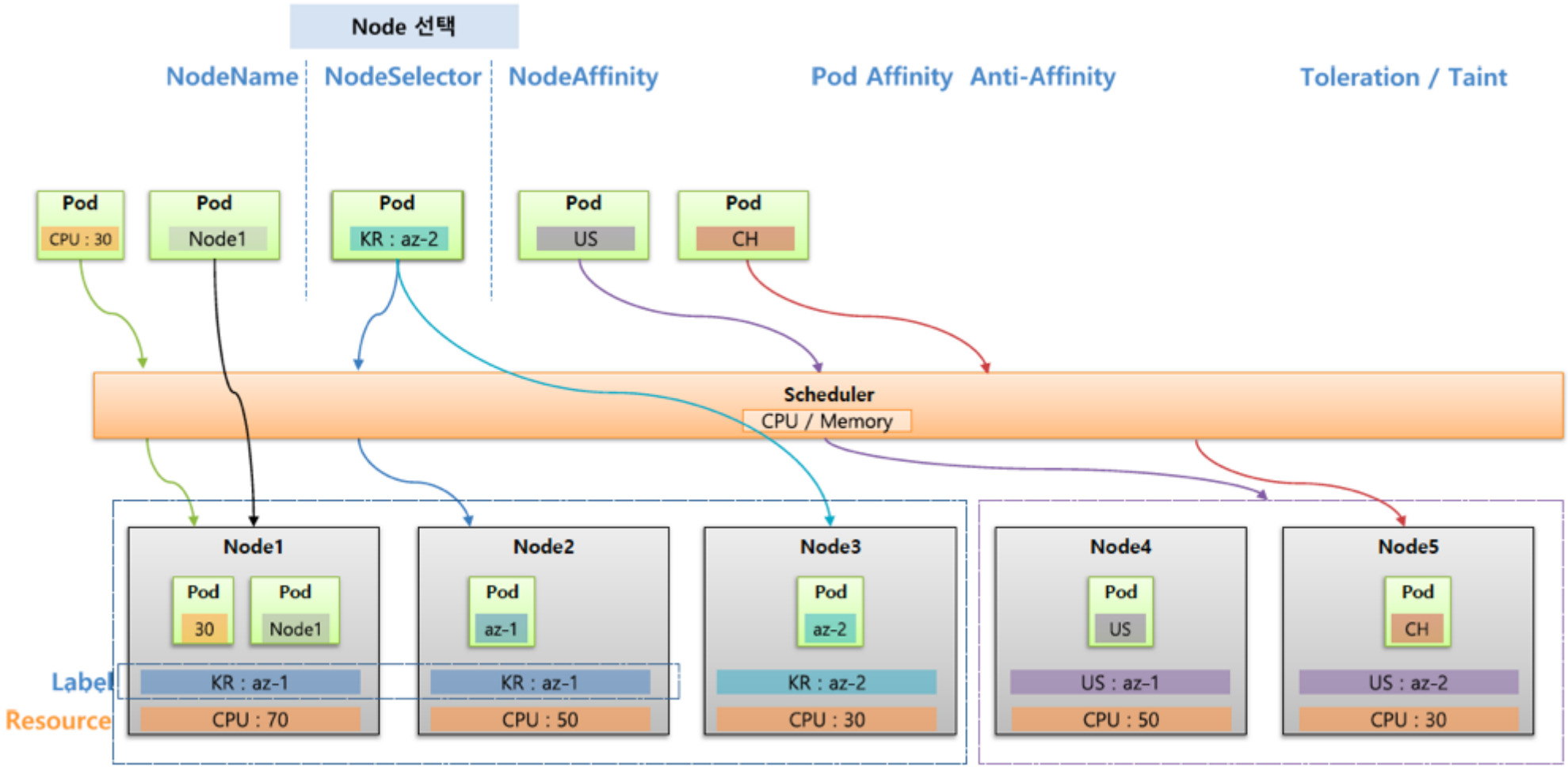
- Stateful App
- 한 Controller에 HPA와 함께 사용안됨

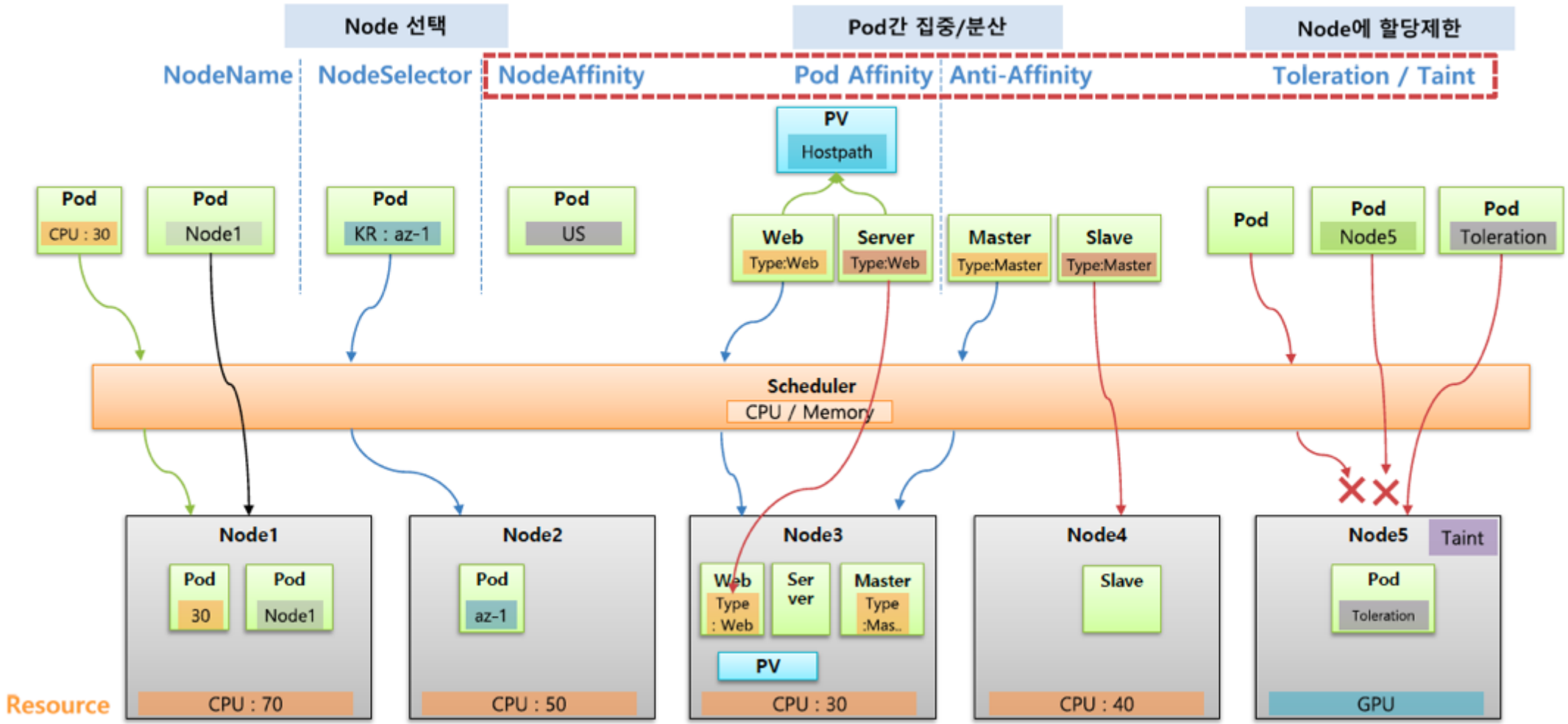
CA (Cluster Autoscaler)

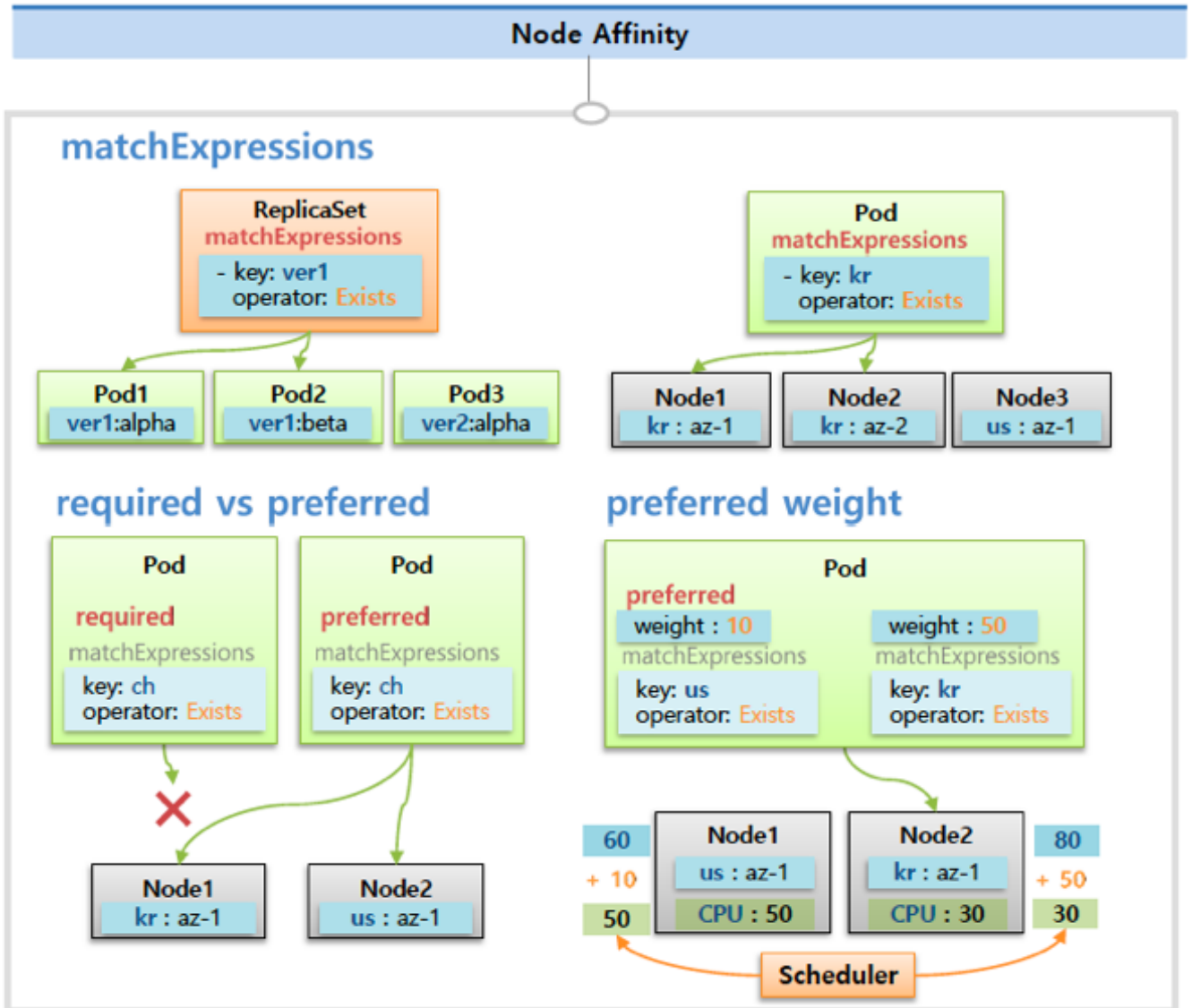


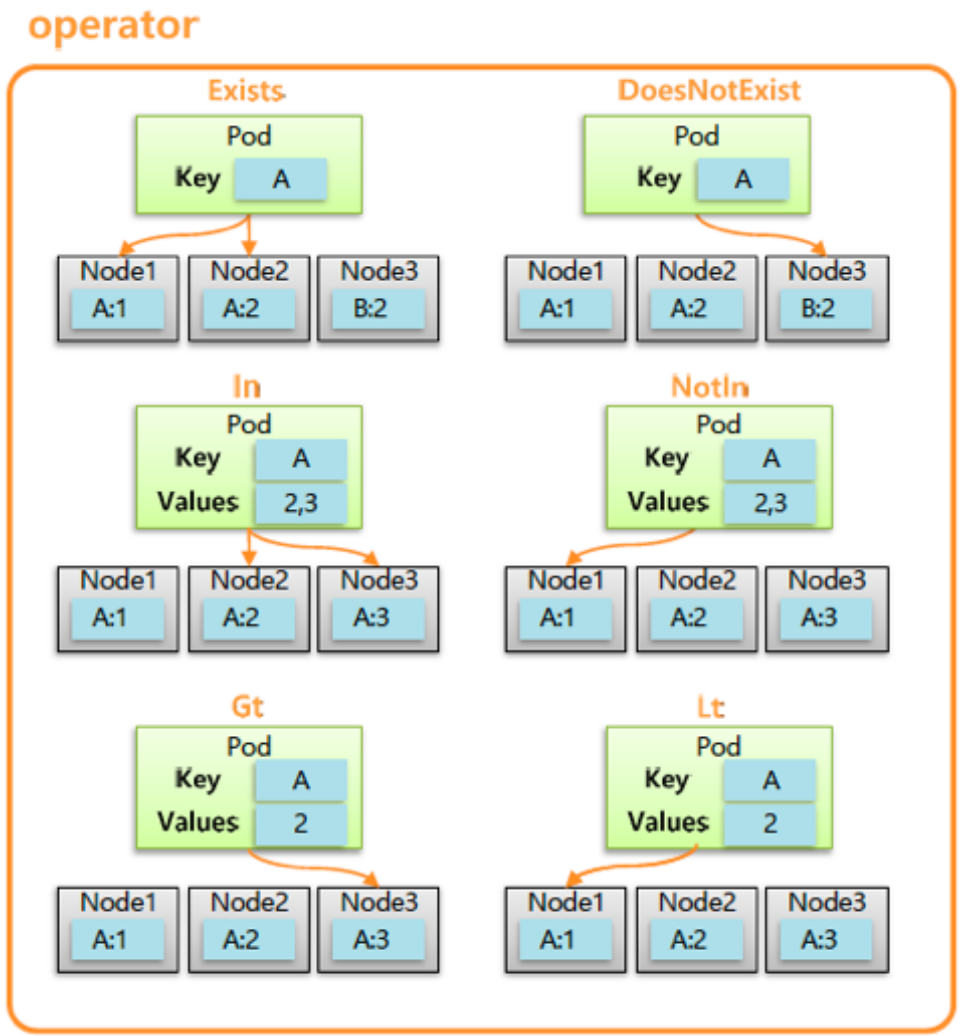

kubectl top
Control Plane Component
Replicas : 1 -> 2
Master
Pod
Controller Manager
Deployment
ReplicaSet
DaemonSet
HPA
VPA
CA
...
..
Create
Pod
Get Memory, CPU (Default 15 Sec)
Pod
kube-apiserver
Resource API
Custom API
External API
Worker Node Component
AddOn Component
Node1
kubelet
Create
Container
Controller Runtime
Docker
rkt
CoreOS
Memory
CPU
Resource Estimator
cAdvisor
Container
Container
Pod
metrics-server
Memory
CPU
Pod
Prometheus
Node2
kubelet

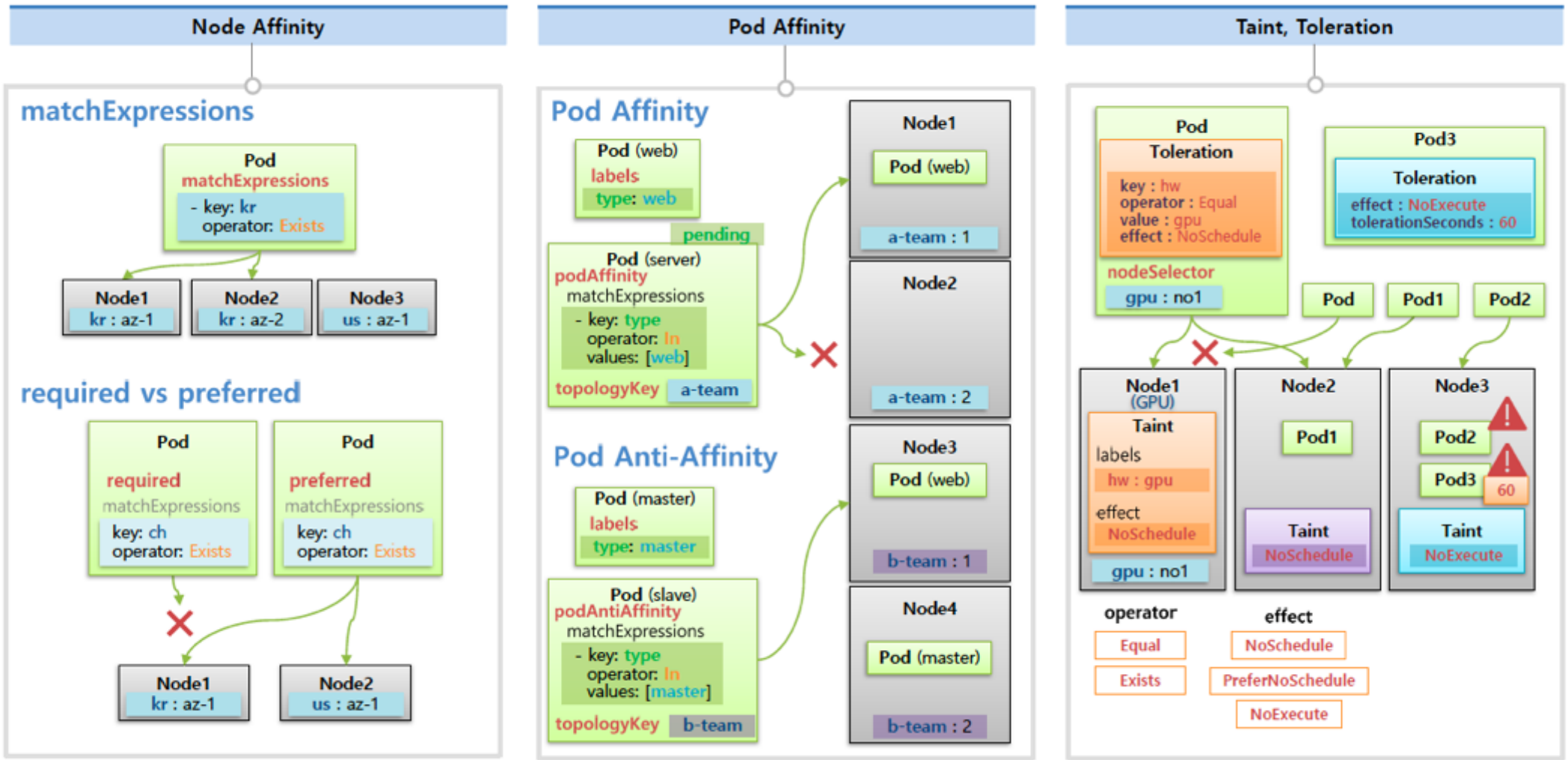

HPA
HPA
target
Deployment
(stateless-app1)
maxRepicas 10
minRepicas 2
metrics
type Resource
name cpu memory
target
type Utilization
averageUtilization 50
type AverageValue
averageValue 100m
type Value
value 10
Deployment
(stateless-app1)
replicas
2
ReplicaSet
Service
Ingress
(nginx-ingress1)
Pod
resources
limits
cpu 500m
requests
cpu 200m
Pod
resources
limits
cpu 500m
requests
cpu 200m
metrics
type : Pods
metrics
type : Object
Custom API
average
target
200m
50%
100m
Scale Out
5 Min
Scale In
average
current
replicas
average
target
desired
replicas
cpu 300m
2
300m
100m
6
cpu 50m
6
50m
100m
3
cpu 50m
3
50m
100m
2
minRepicas


Node 선택
NodeName
NodeSelector
NodeAffinity
Pod Affinity
Anti-Affinity
Toleration / Taint
Pod
CPU : 30
Pod
Node1
Pod
KR : az-2
Pod
US
Pod
CH
Scheduler
CPU / Memory
Node1
Pod
30
Pod
Node1
KR : az-1
CPU : 70
Node2
Pod
az-1
KR : az-1
CPU : 50
Node3
Pod
az-2
KR : az-2
CPU : 30
Node4
Pod
US
US : az-1
CPU : 50
Node5
Pod
CH
US : az-2
CPU : 30
Label
Resource

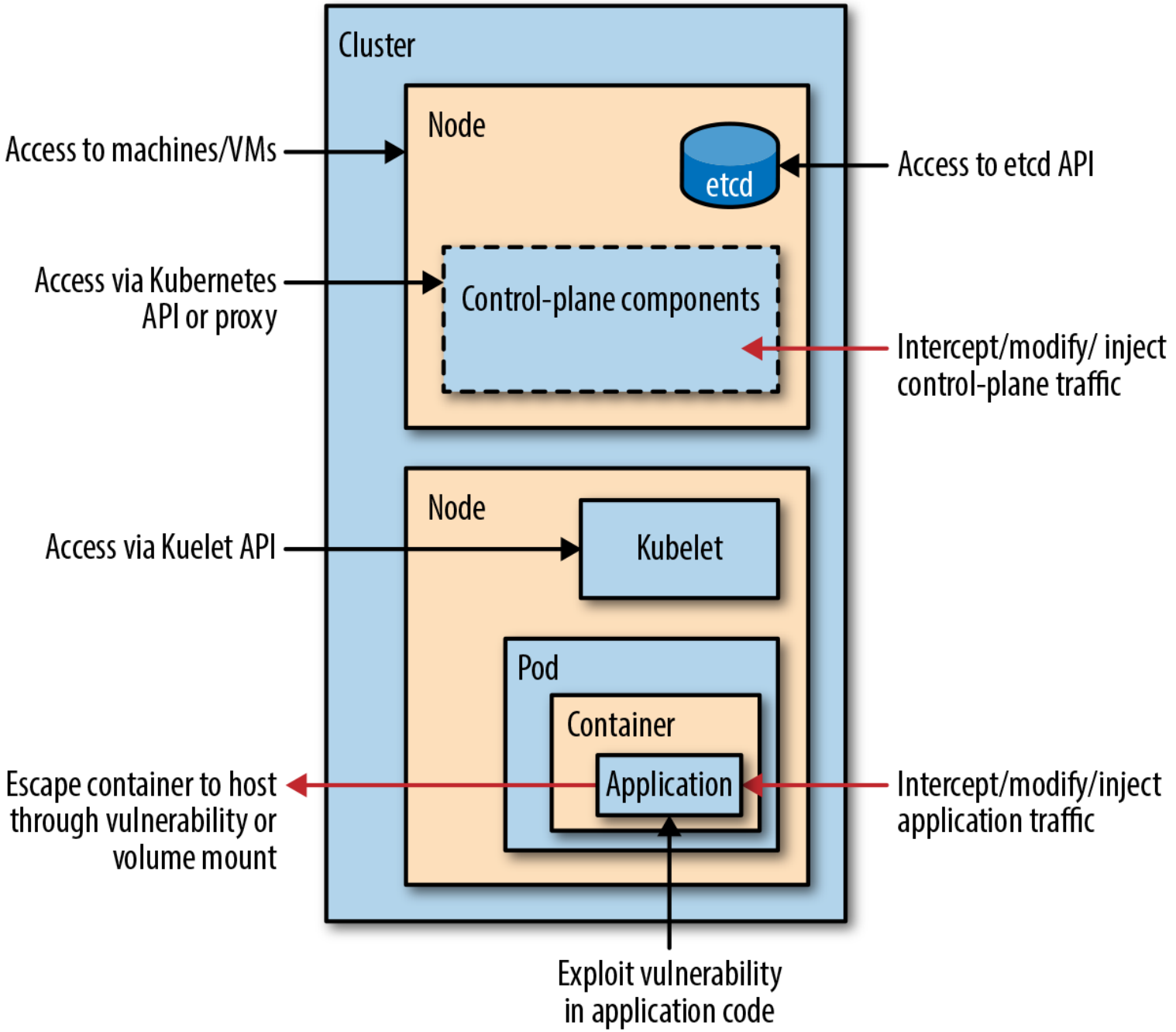

Node 선택
Pod간 집중/분산
Node에 할당제한
NodeName
NodeSelector
NodeAffinity
Pod Affinity
Anti-Affinity
Toleration / Taint
PV
Hostpath
Pod
CPU : 30
Pod
Node1
Pod
KR : az-1
Pod
US
Web
Type:Web
Server
Type:Web
Master
Type:Master
Slave
Type:Master
Pod
Pod
Node5
Pod
Toleration
Scheduler
CPU / Memory
Node1
Pod
30
Pod
Node1
CPU : 70
Node2
Pod
az-1
CPU : 50
Node3
Web
Type
: Web
Ser
ver
Master
Type
:Mas..
PV
CPU : 30
Node4
Slave
CPU : 40
Node5
Taint
Pod
Toleration
GPU
Resource


Node Affinity
matchExpressions
ReplicaSet
matchExpressions
- key: ver1
operator: Exists
Pod1
ver1:alpha
Pod2
ver1:beta
Pod3
ver2:alpha
Pod
matchExpressions
- key: kr
operator: Exists
Node1
kr : az-1
Node2
kr : az-2
Node3
us : az-1
required vs preferred
Pod
required
matchExpressions
key: ch
operator: Exists
Pod
preferred
matchExpressions
key: ch
operator: Exists
Node1
kr : az-1
Node2
us : az-1
preferred weight
Pod
preferred
weight : 10
matchExpressions
key: us
operator: Exists
weight : 50
matchExpressions
key: kr
operator: Exists
60
+ 10
50
Node1
us : az-1
CPU : 50
Node2
kr : az-1
CPU : 30
80
+ 50
30
Scheduler


operator
Exists
Pod
Key A
Node1 A:1
Node2 A:2
Node3 B:2
DoesNotExist
Pod
Key A
Node1 A:1
Node2 A:2
Node3 B:2
In
Pod
Key A
Values 2,3
Node1 A:1
Node2 A:2
Node3 A:3
NotIn
Pod
Key A
Values 2,3
Node1 A:1
Node2 A:2
Node3 A:3
Gt
Pod
Key A
Values 2
Node1 A:1
Node2 A:2
Node3 A:3
Lt
Pod
Key A
Values 2
Node1 A:1
Node2 A:2
Node3 A:3

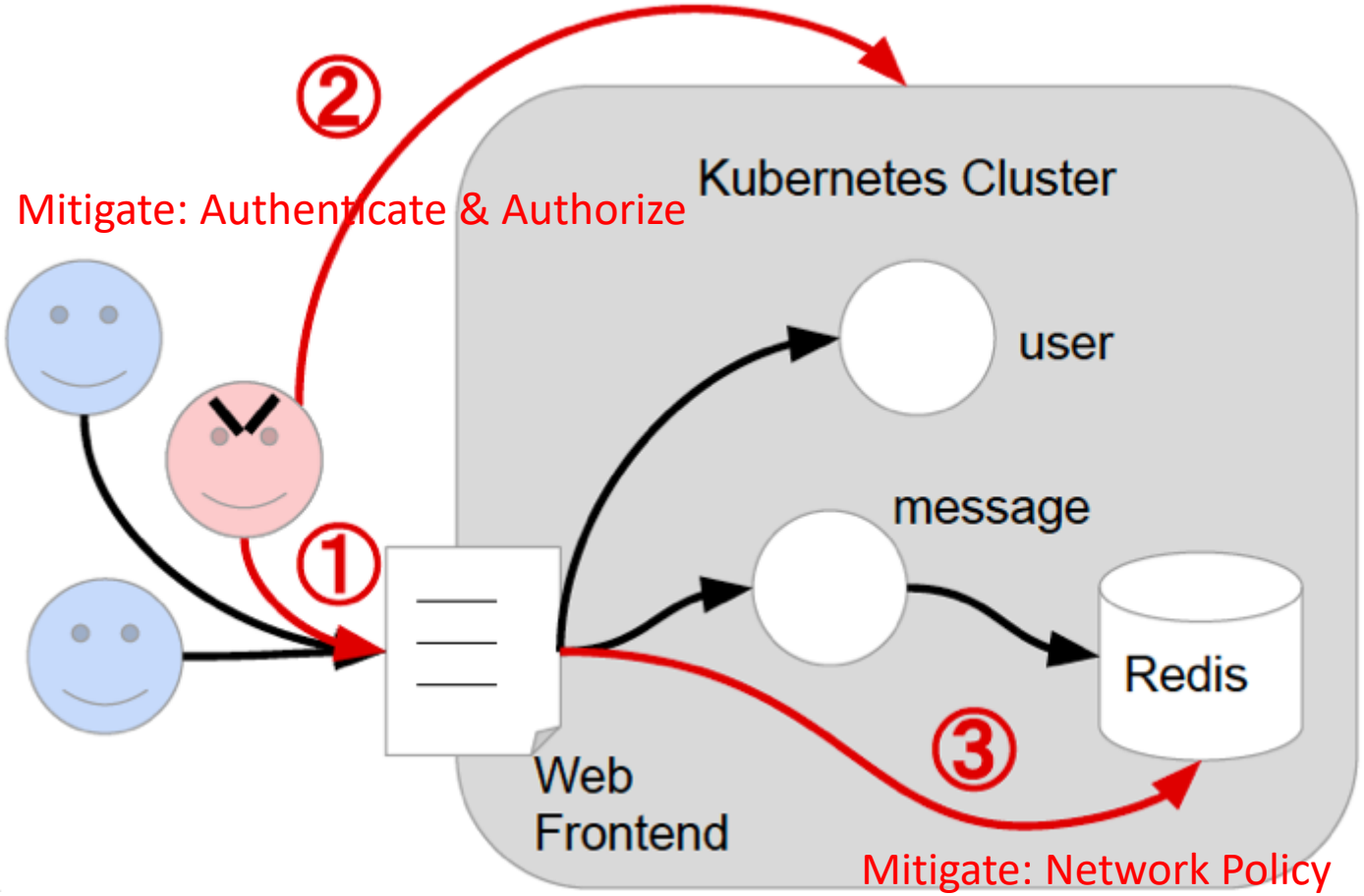

Node Affinity
matchExpressions
Pod
matchExpressions
- key: kr
operator: Exists
Node1
kr : az-1
Node2
kr : az-2
Node3
us : az-1
required vs preferred
Pod
required
matchExpressions
key: ch
operator: Exists
Pod
preferred
matchExpressions
key: ch
operator: Exists
Node1
kr : az-1
Node2
us : az-1
Pod Affinity
Pod Affinity
Pod (web)
labels
type: web
pending
Pod (server)
podAffinity
matchExpressions
- key: type
operator: In
values: [web]
topologyKey
a-team
Node1
Pod (web)
a-team : 1
Node2
a-team : 2
Pod Anti-Affinity
Pod (master)
labels
type: master
Pod (slave)
podAntiAffinity
matchExpressions
- key: type
operator: In
values: [master]
topologyKey
b-team
Node3
Pod (web)
b-team : 1
Node4
Pod (master)
b-team : 2
Taint, Toleration
Pod
Toleration
key : hw
operator : Equal
value : gpu
effect : NoSchedule
nodeSelector
gpu : no1
Pod3
Toleration
effect : NoExecute
tolerationSeconds : 60
Pod
Pod1
Pod2
Node1
(GPU)
Taint
labels
hw : gpu
effect
NoSchedule
gpu : no1
Node2
Pod1
Taint
NoSchedule
Node3
Pod2
Pod3
60
Taint
NoExecute
operator
Equal
Exists
effect
NoSchedule
PreferNoSchedule
NoExecute

Attack Surface



Attack Surface: API Server

1. Get token from frontend Pod
2. Use token to attack cluster API server
3. Get secrets etc. to further attack



Authentication & Authorization

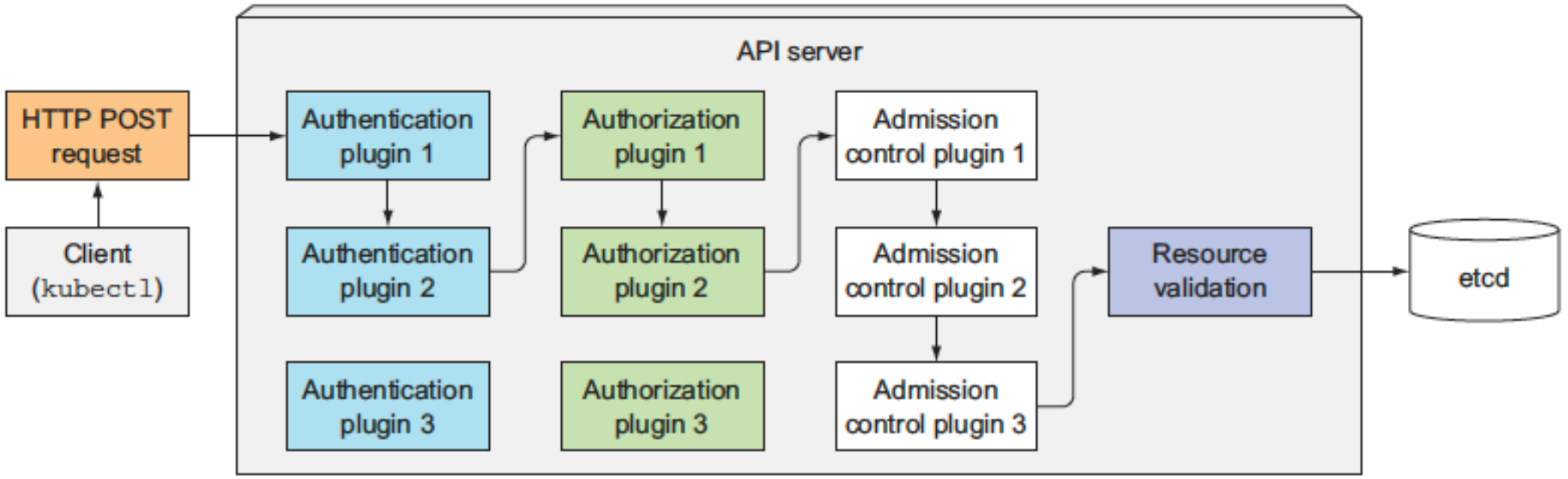
Authentication

Who you are

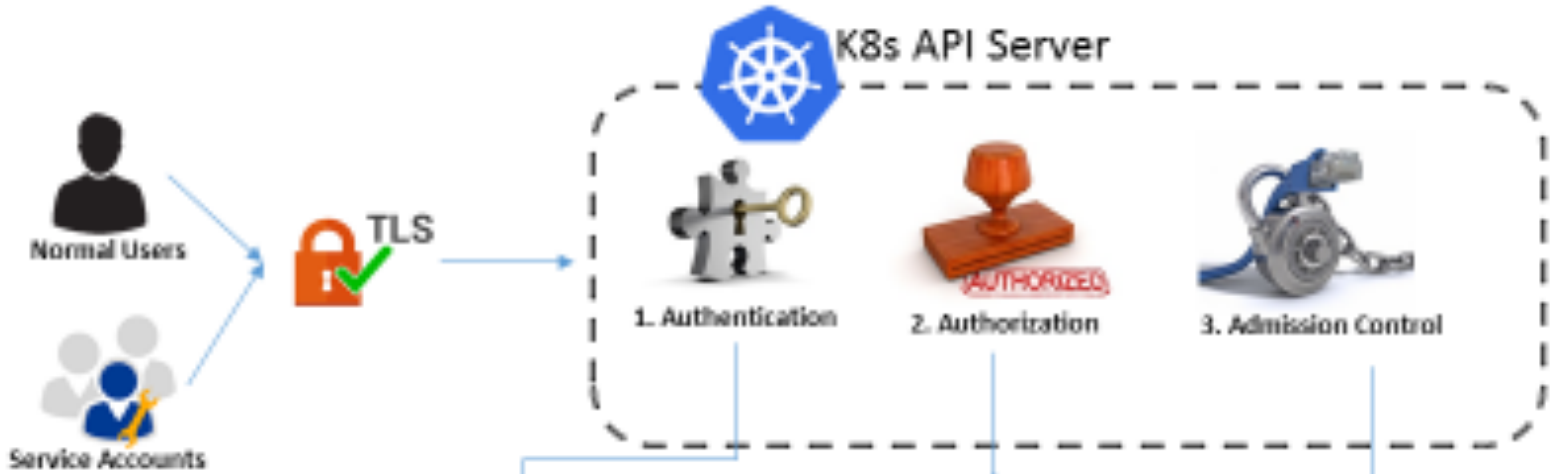
Authorization

What you can do

Authentication & Authorization in API Server



Accessing the Kubernetes API



1. X509 Client Certs
2. Static Token File
3. Bootstrap Tokens
4. Static Password File
5. Service Account Tokens
6. OpenID Connect Tokens
7. Webhook Token authentication
8. Authenticating Proxy
9. Anonymous
10. User Impersonation

- Authorization Modules:

1. Node
2. ABAC
3. RBAC
4. RBAC
5. Webhook

- Using Flags for your authorization Module
- Privilege escalation visa pod creation.

1. AlwaysPullImages
2. DefaultStorageClass
3. DefaultTolerationSeconds
4. DenyExecOnPrivileges (Depr)
5. DenyEscalatingExec
6. EventRateLimit
7. ExtendedResourceToleraton
8. ImagePolicyWebhook
9. Initializers
10. LimitPodHardAntiAffinityTopology
11. LimitRanger
12. MutatingAdmissionWebhook
13. NamespaceAutoProvision
14. NamespaceExists
15. NamespaceLifecycle
16. NodeRestriction
17. OwnerReferencesPermissionEnforcement
18. PersistentVolumeLabel (Depr)
19. PodNodeSelector
20. PersistentVolumeClaimResize
21. PodPreset
22. PodSecurityPolicy
23. PodTolerationRestriction
24. Priority
25. ResourceQuota
26. SecurityContextDeny
27. ValidatingAdmissionWebhook

Kubernetes Authentication

Mechanism	Secret Source	Usage
X509 Client Certs	CSR generated externally and signed with the cluster CA key	Enterprise CA / PKI
	Via Kubernetes API CertificateSigningRequest	Kubernetes cluster admin
Bearer token	Bootstrap token	Internal use
	Node authentication token	Internal use
	Static token file	Insecure
	ServiceAccount token	Pods, containers, applications, *users*
	OIDC token	Users
HTTP Basic auth	Static password file	Insecure
Auth proxy	N/A (trust proxy)	Integration
Impersonate	N/A (trust account)	Integration and administration

Certificate Based Authentication

- Kubernetes is configured with a Certificate Authority (CA) 

`/etc/kubernetes/pki/ca.crt` — Public certificate

`/etc/kubernetes/pki/ca.key` — Private key 

- Every SSL certificate signed with this CA will be accepted by the Kubernetes API
- Possible options for creating certificates: OpenSSL or CloudFlare's PKI toolkit
- Two important fields in the SSL certificate:
  - Common Name (CN): Kubernetes will interpret this value as the **user**
  - Organization (O): Kubernetes will interpret this value as the **group**

TLS Certificate

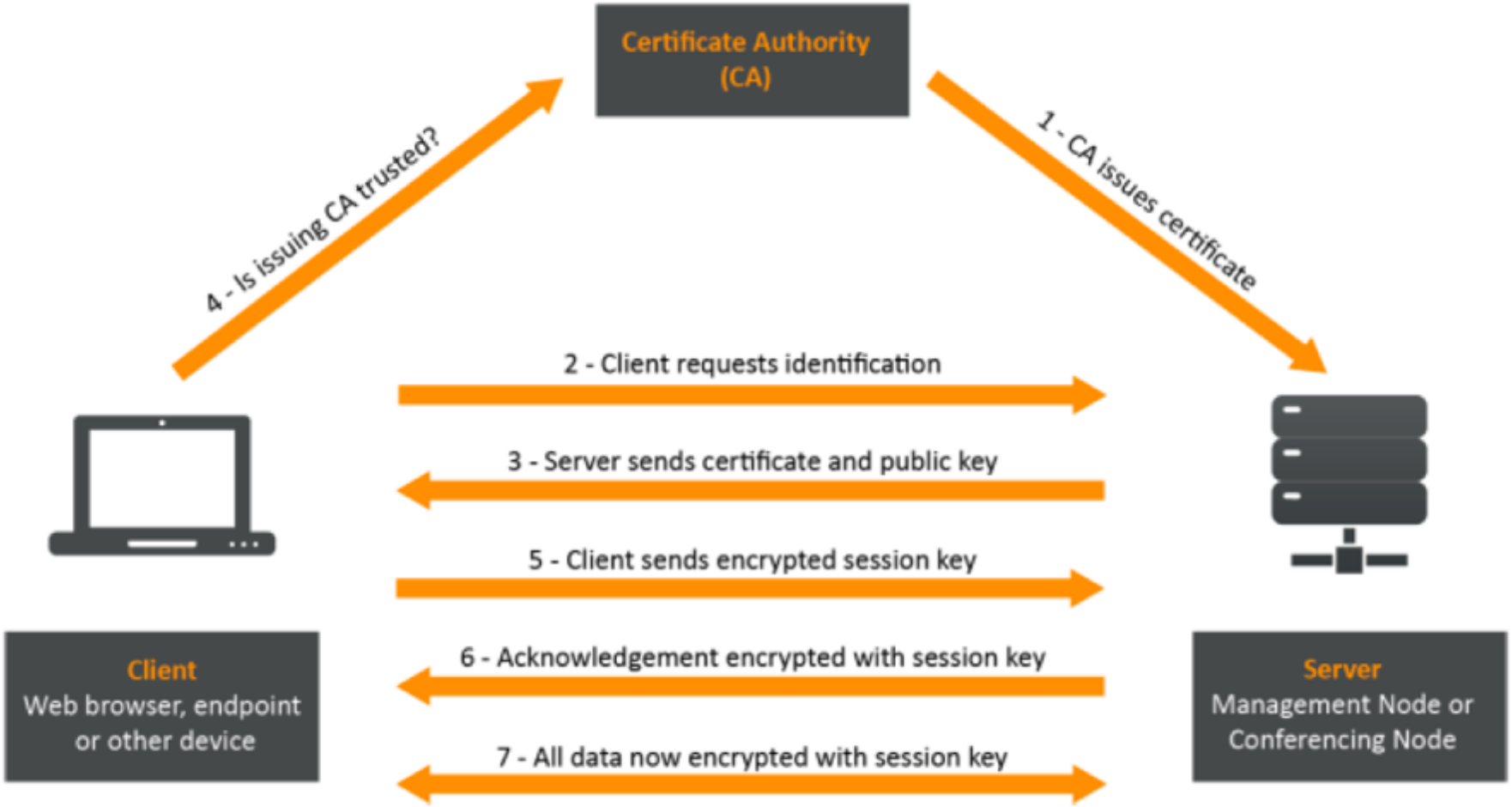


그림 출처: https://docs.pexip.com/admin/certificate_management.htm

Certificate & CA in Kubernetes

CERTIFICATE	RESIDUAL TIME	Certificate Path	CERTIFICATE AUTHORITY
admin.conf	365d	/etc/kubernetes	
apiserver	365d	/etc/kubernetes/pki	ca
apiserver-etcd-client	365d	/etc/kubernetes/pki/etcd	etcd-ca
apiserver-kubelet-client	365d	/etc/kubernetes/pki	ca
controller-manager.conf	365d	/etc/kubernetes/pki	
etcd-healthcheck-client	365d	/etc/kubernetes/pki/etcd	etcd-ca
etcd-peer	365d	/etc/kubernetes/pki/etcd	etcd-ca
etcd-server	365d	/etc/kubernetes/pki/etcd	etcd-ca
front-proxy-client	365d	/etc/kubernetes/pki	front-proxy-ca
scheduler.conf	365d	/etc/kubernetes	

CERTIFICATE AUTHORITY	RESIDUAL TIME	Certificate Path
ca	10y	/etc/kubernetes/pki
etcd-ca	10y	/etc/kubernetes/pki/etcd
front-proxy-ca	10y	/etc/kubernetes/pki

참고: front-proxy 인증서는 kube-proxy에서 API 서버 확장을 지원할 때만 kube-proxy에서 필요하다.

Creating User Certificate (1)

- Create private key (if it does not exist)

```
openssl genrsa -out juan.key 2048
```

- Create certificate signing request (CSR)

```
openssl req -new -key juan.key -out juan.csr -subj "/CN=juan/O=devs"
```

user group

- Send the CSR to the administrator

- Create certificate from CSR using the cluster authority

```
openssl x509 -req -in juan.csr -CA CA_LOCATION/ca.crt -CAkey
CA_LOCATION/ca.key -CAcreateserial -out juan.crt -days 500
```

Creating User Certificate (2)

- To add in your local machine the new configuration:
  - Download the cluster authority and generated certificate
  - Add the new cluster to kubectl

```
kubectl config set-cluster sandbox --certificate-authority=ca.pem
--embed-certs=true --server=https://<PUBLIC_ADDRESS_OF_YOUR_CLUSTER>:6443
```

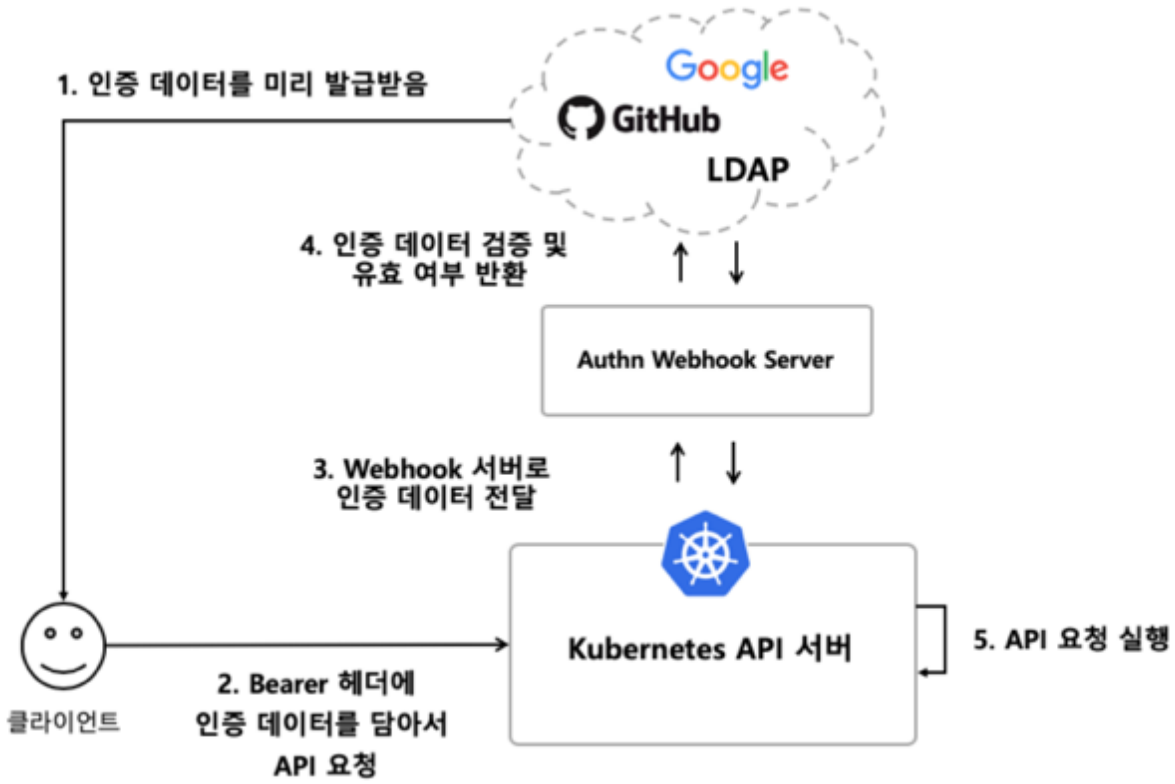
  - Add the new credentials to kubectl

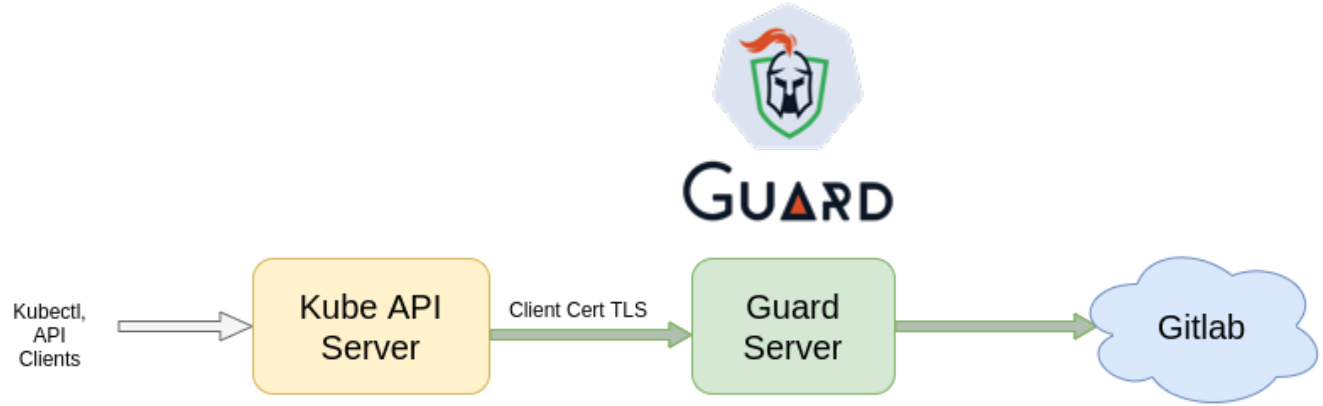
```
kubectl config set-credentials juan --client-certificate=juan.crt
--client-key=juan.key --embed-certs=true
```

  - Add the new context to kubectl

```
kubectl config set-context sandbox-juan --cluster=sandbox --user=juan
```

Token Webhook






```
{
  "apiVersion": "authentication.k8s.io/v1beta1",
  "kind": "TokenReview",
  "status": {
    "authenticated": true,
    "user": {
      "username": "<gitlab-login>",
      "uid": "<gitlab-id>",
      "groups": [
        "<group-1>",
        "<group-2>"
      ]
    }
  }
}
```

Token file, Gitlab, Google, Azure, LDAP, EKS

dex

Kubernetes Authorization

Mechanism	Decision source	Usage
Node	API Server built-in	Internal use (kubelets)
ABAC	Static file	Insecure, deprecated
RBAC	API Objects	Users and administrators
WebHook	External services	Integration
AlwaysDeny AlwaysAllow	API Server built-in	Testing

RBAC

Subjects

API Resources

Operations (Verbs)

Role

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   namespace: test
   name: pod-access
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list"]
```

WHICH RESOURCES

WHICH OPERATIONS

- Need to specify:
  - Api group
  - Name

Find it in the API reference, examples

Group	Version	Kind
apps	v1	Deployment

Group	Version	Kind
core	v1	Pod

When it is core, we use an empty string

API Group

API Group	API Objects
rbac.authorization.k8s.io/v1	ClusterRole Role ClusterRoleBinding RoleBinding
authentication.k8s.io/v1	TokenReview
admissionregistration.k8s.io/v1	MutatingWebhookConfiguration ValidatingWebhookConfiguration
authorization.k8s.io/v1	LocalSubjectAccessReview SelfSubjectAccessReview SelfSubjectRulesReview SubjectAccessReview
certificates.k8s.io/v1beta1	CertificateSigningRequest
policy/v1beta1	PodSecurityPolicy

Role Binding

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: devs-read-pods
  namespace: test
subjects:
- kind: Group
  name: devs
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-access
  apiGroup: rbac.authorization.k8s.io
```

- Examples
  - User
  - Group
  - ...

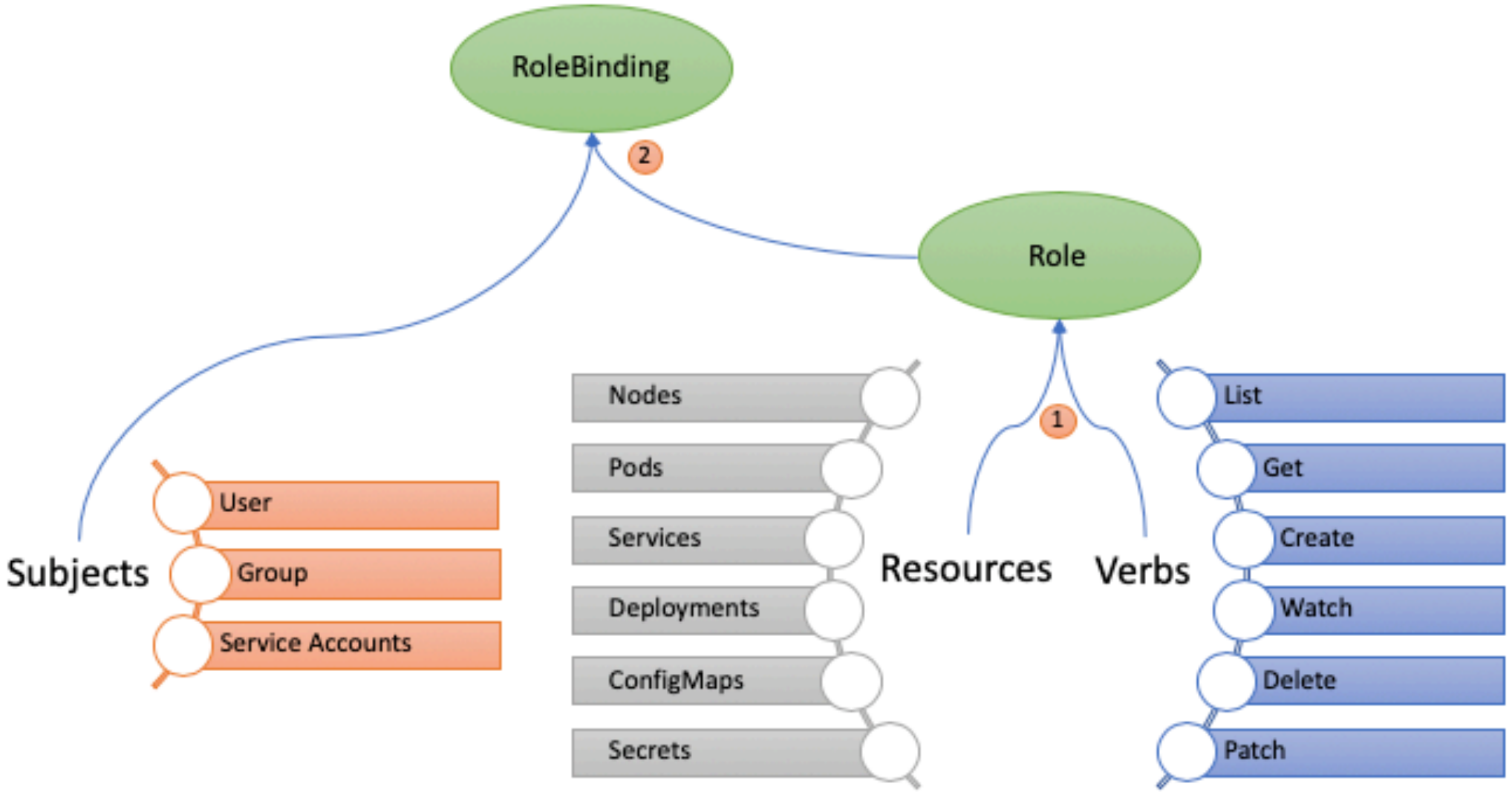
Later we will see another one

WHICH SUBJECTS

Used to specify which api group the kind belongs to

WHICH ROLE (ONLY ONE PER BINDING)

Role & Role Binding



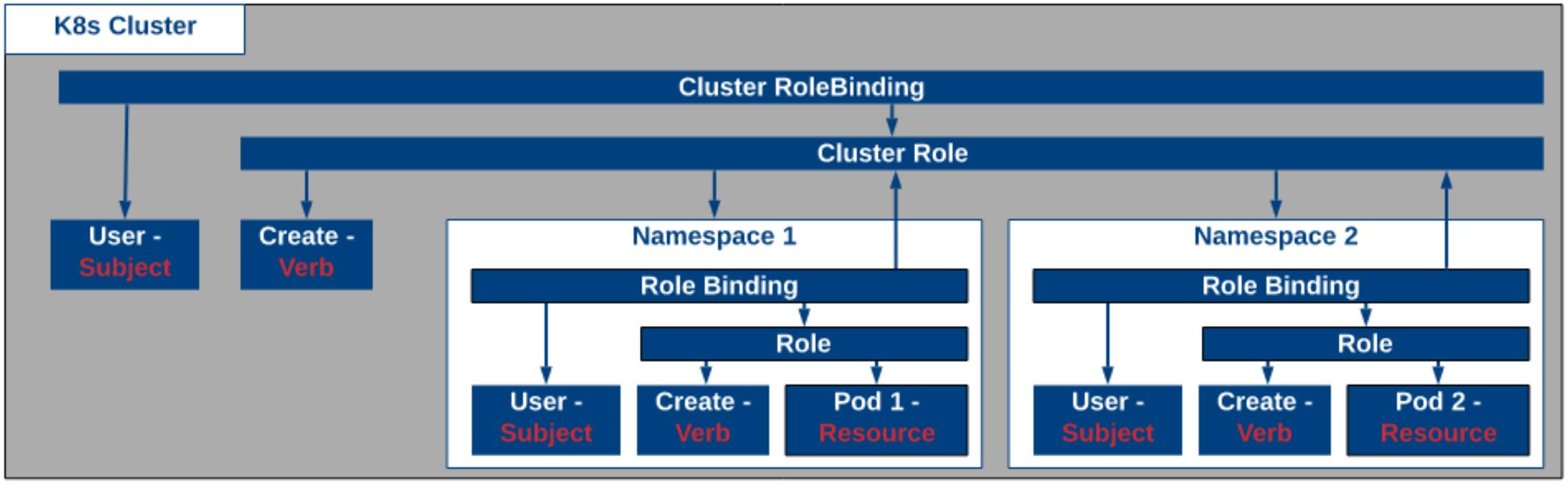
Cluster Role

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   name: pod-access
   namespace: test
rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list"]
```

Only difference

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
   name: all-pod-access

rules:
 - apiGroups: [""]
   resources: ["pods"]
   verbs: ["get", "list"]
```

Role & Cluster Role



Cluster Role Binding

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: devs-read-pods
  namespace: test
subjects:
- kind: User
  name: jsalmeron # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: ns-admin
  apiGroup: rbac.authorization.k8s.io
```

Only differences

Only differences

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: salme-reads-all-pods

subjects:
- kind: User
  name: jsalmeron # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: all-pod-access
  apiGroup: rbac.authorization.k8s.io
```

Default Cluster Role Bindings

- Kubernetes includes some ClusterRoleBindings. For example:
  - **system:basic-user**: For unauthenticated users (group **system:unauthenticated**). No operations are allowed.
  - **cluster-admin:** For members of the **system:masters** group. Can do any operation on the cluster (using **cluster-admin** ClusterRole).

Admin accounts can be created belonging to this group

```
openssl req ... -subj "/CN=dbarranco/O=system:masters"
```

  - ClusterRoleBindings for the **different components** of the cluster (kube-controller-manager, kube-scheduler, kube-proxy …)

Kubernetes RBAC API Discovery Roles

Default ClusterRole	Default ClusterRoleBinding	Description
system:basic-user	**system:authenticated** group	Allows a user read-only access to basic information about themselves. Prior to v1.14, this role was also bound to `system:unauthenticated` by default.
system:discovery	**system:authenticated** group	Allows read-only access to API discovery endpoints needed to discover and negotiate an API level. Prior to v1.14, this role was also bound to `system:unauthenticated` by default.
system:public-info-viewer	**system:authenticated** and **system:unauthenticated** groups	Allows read-only access to non-sensitive information about the cluster. Introduced in Kubernetes v1.14.

Kubernetes RBAC API discovery roles

Default ClusterRole	Default ClusterRoleBinding	Description
cluster-admin	**system:masters** group	Allows super-user access to perform any action on any resource. When used in a **ClusterRoleBinding**, it gives full control over every resource in the cluster and in all namespaces. When used in a **RoleBinding**, it gives full control over every resource in the role binding's namespace, including the namespace itself.
admin	None	Allows admin access, intended to be granted within a namespace using a **RoleBinding**. If used in a **RoleBinding**, allows read/write access to most resources in a namespace, including the ability to create roles and role bindings within the namespace. This role does not allow write access to resource quota or to the namespace itself.
edit	None	Allows read/write access to most objects in a namespace. This role does not allow viewing or modifying roles or role bindings. However, this role allows accessing Secrets and running Pods as any ServiceAccount in the namespace, so it can be used to gain the API access levels of any ServiceAccount in the namespace.
view	None	Allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings. This role does not allow viewing Secrets, since reading the contents of Secrets enables access to ServiceAccount credentials in the namespace, which would allow API access as any ServiceAccount in the namespace (a form of privilege escalation).

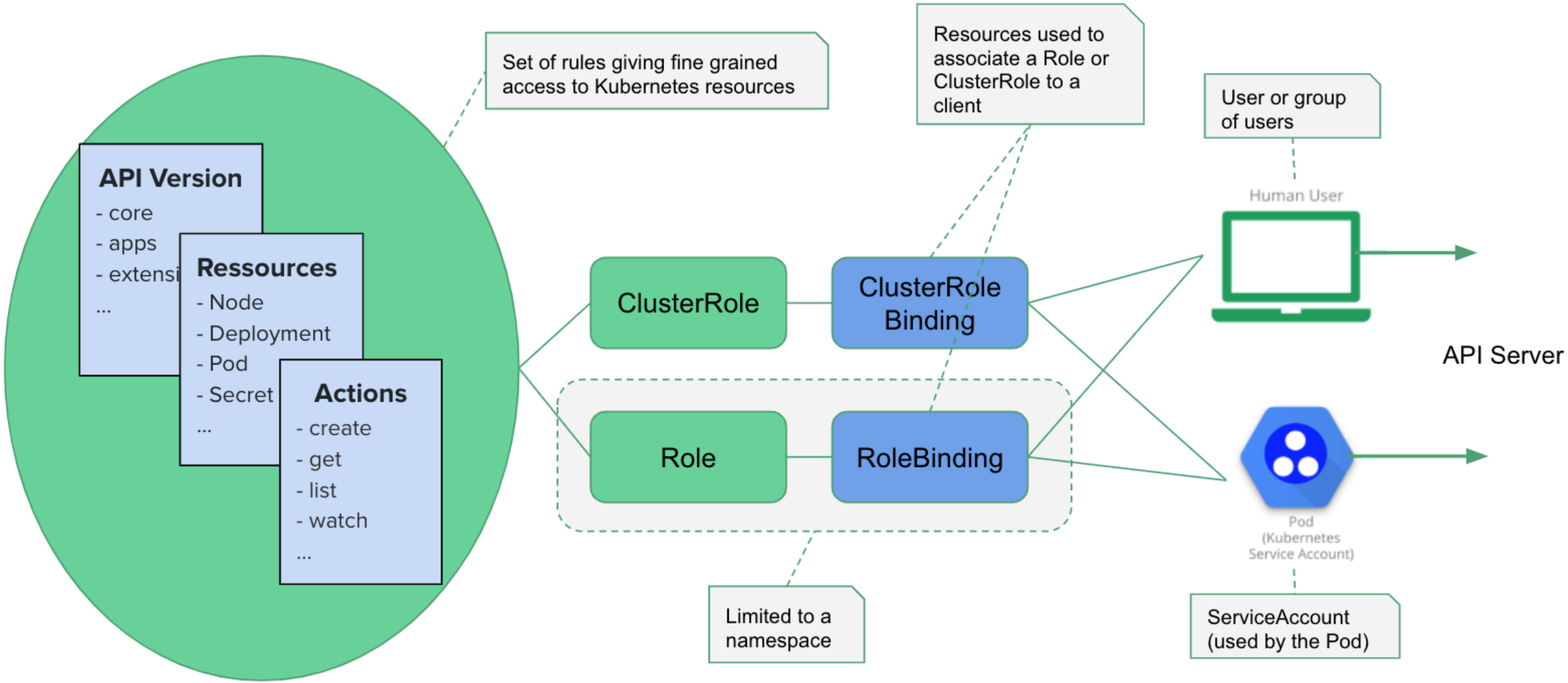
Core component roles

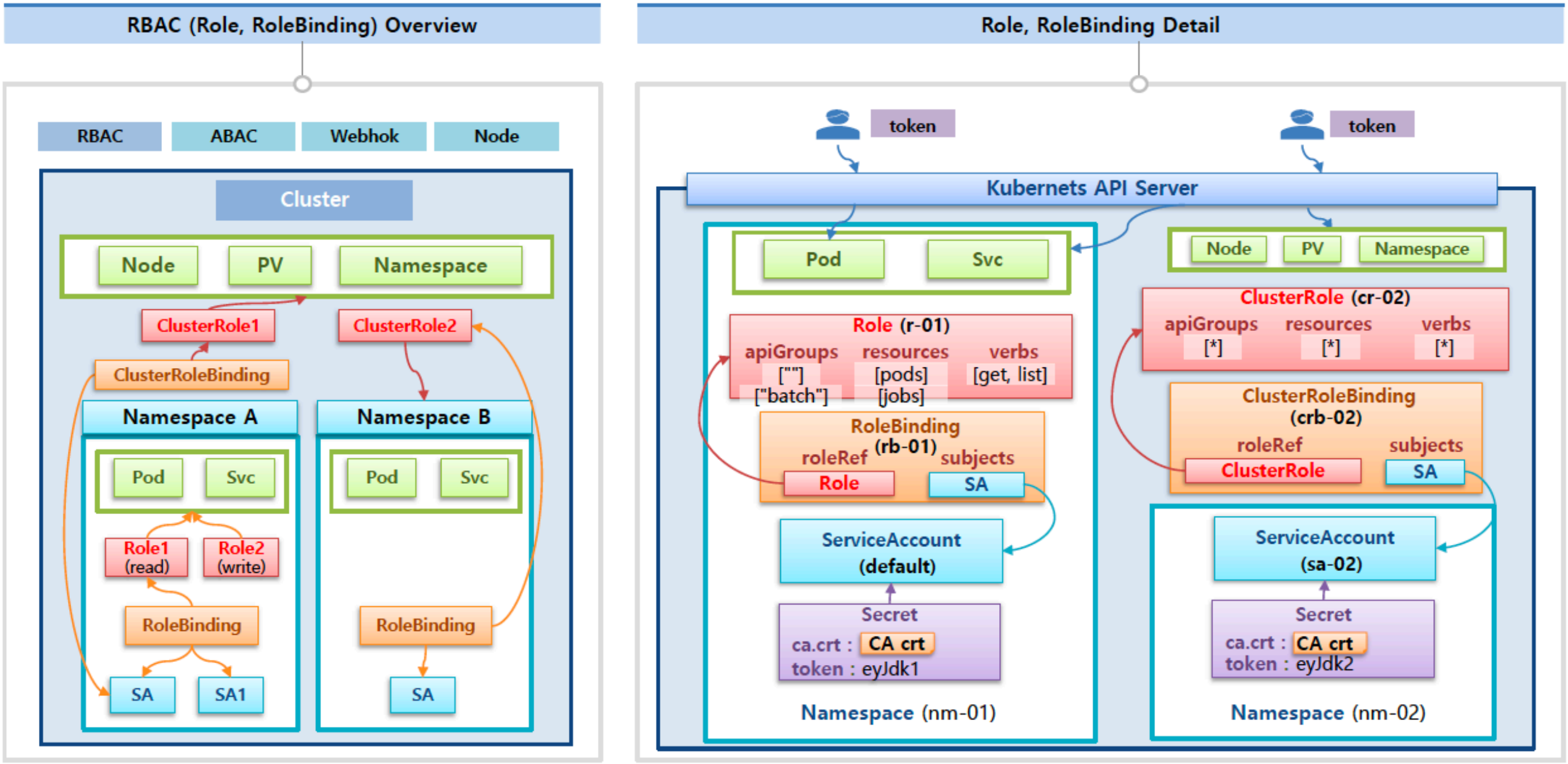
Default ClusterRole	Default ClusterRoleBinding	Description
system:kube-scheduler	**system:kube-scheduler** user	Allows access to the resources required by the scheduler component.
system:volume-scheduler	**system:kube-scheduler** user	Allows access to the volume resources required by the kube-scheduler component.
system:kube-controller-manager	**system:kube-controller-manager** user	Allows access to the resources required by the controller manager component. The permissions required by individual controllers are detailed in the controller roles.
system:node	None	Allows access to resources required by the kubelet, **including read access to all secrets, and write access to all pod status objects**. You should use the Node authorizer and NodeRestriction admission plugin instead of the `system:node` role, and allow granting API access to kubelets based on the Pods scheduled to run on them. The `system:node` role only exists for compatibility with Kubernetes clusters upgraded from versions prior to v1.8.
system:node-proxier	**system:kube-proxy** user	Allows access to the resources required by the kube-proxy component.

Other component roles

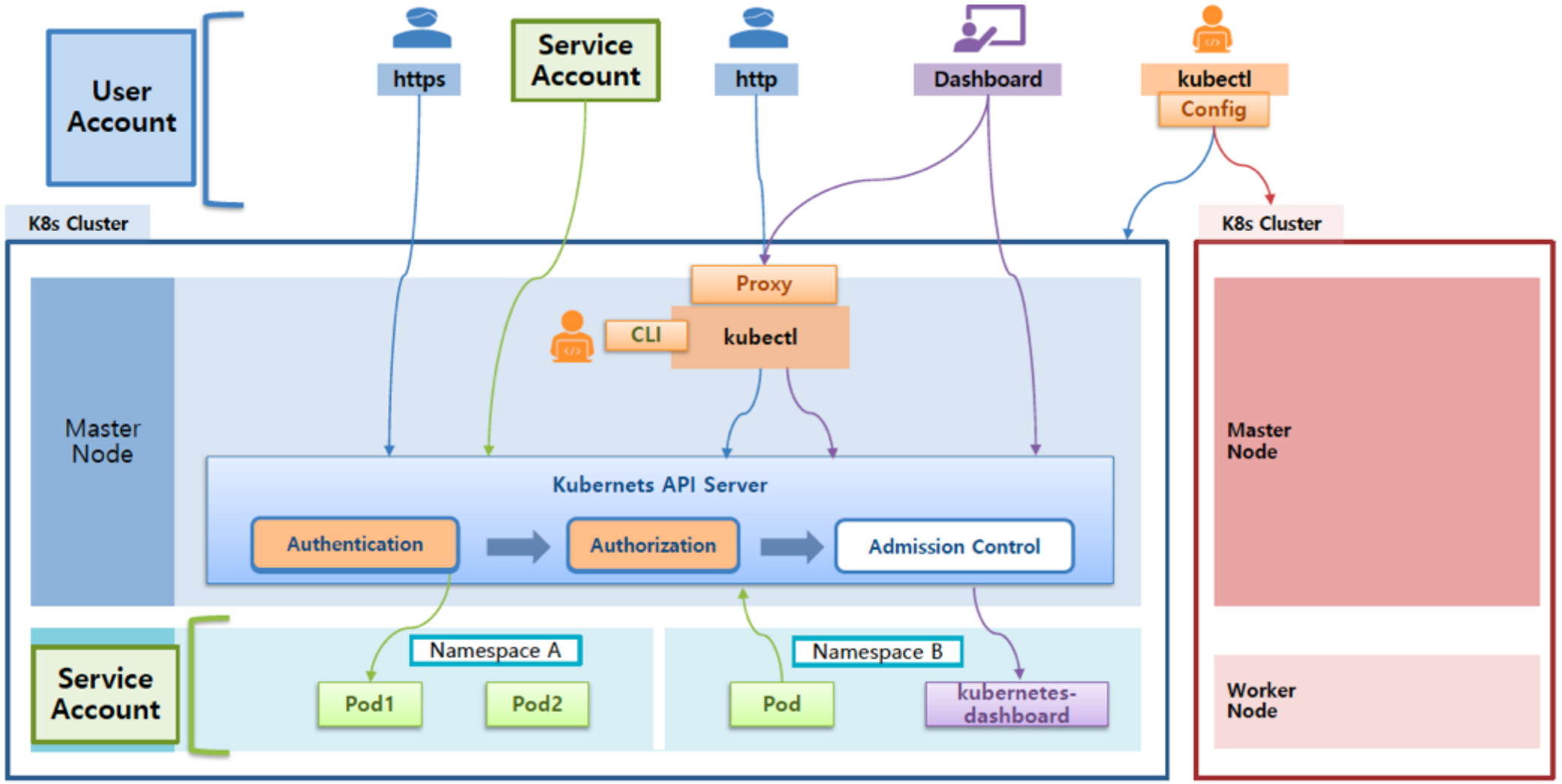
Default ClusterRole	Default ClusterRoleBinding	Description
system:auth-delegator	None	Allows delegated authentication and authorization checks. This is commonly used by add-on API servers for unified authentication and authorization.
system:heapster	None	Role for the Heapster component (deprecated).
system:kube-aggregator	None	Role for the kube-aggregator component.
system:kube-dns	**kube-dns** service account in the **kube-system** namespace	Role for the kube-dns component.
system:kubelet-api-admin	None	Allows full access to the kubelet API.
system:node-bootstrapper	None	Allows access to the resources required to perform kubelet TLS bootstrapping.
system:node-problem-detector	None	Role for the node-problem-detector component.
system:persistent-volume-provisioner	None	Allows access to the resources required by most dynamic volume provisioners.

Kubernetes RBAC

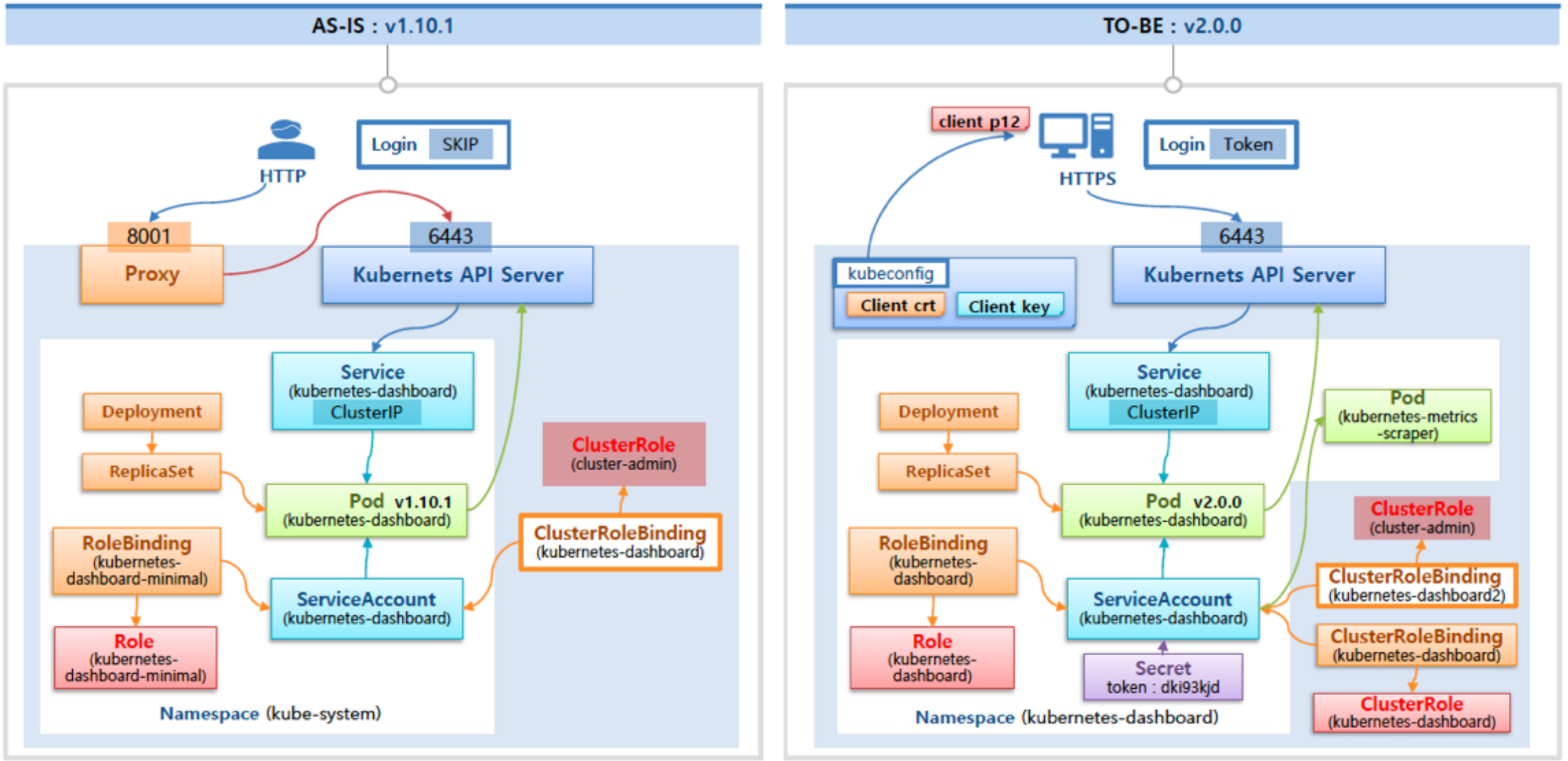



RBAC (Role, RoleBinding) Overview
RBAC
ABAC
Webhok
Node
Cluster
Node
PV
Namespace
ClusterRole1
ClusterRole2
ClusterRoleBinding
Namespace A
Namespace B
Pod
Svc
Pod
Svc
Role1
(read)
Role2
(write)
RoleBinding
RoleBinding
SA
SA1
SA
Role, RoleBinding Detail
token
token
Kubernets API Server
Pod
Svc
Node
PV
Namespace
ClusterRole (cr-02)
apiGroups
[*]
resources
[*]
verbs
[*]
Role (r-01)
apiGroups
[""]
["batch"]
resources
[pods]
[jobs]
verbs
[get, list]
ClusterRoleBinding
(crb-02)
roleRef
ClusterRole
subjects
SA
RoleBinding
(rb-01)
roleRef
Role
subjects
SA
ServiceAccount
(default)
ServiceAccount
(sa-02)
Secret
ca.crt : CA crt
token : eyJdk1
Secret
ca.crt : CA crt
token : eyJdk2
Namespace (nm-01)
Namespace (nm-02)

Service Account



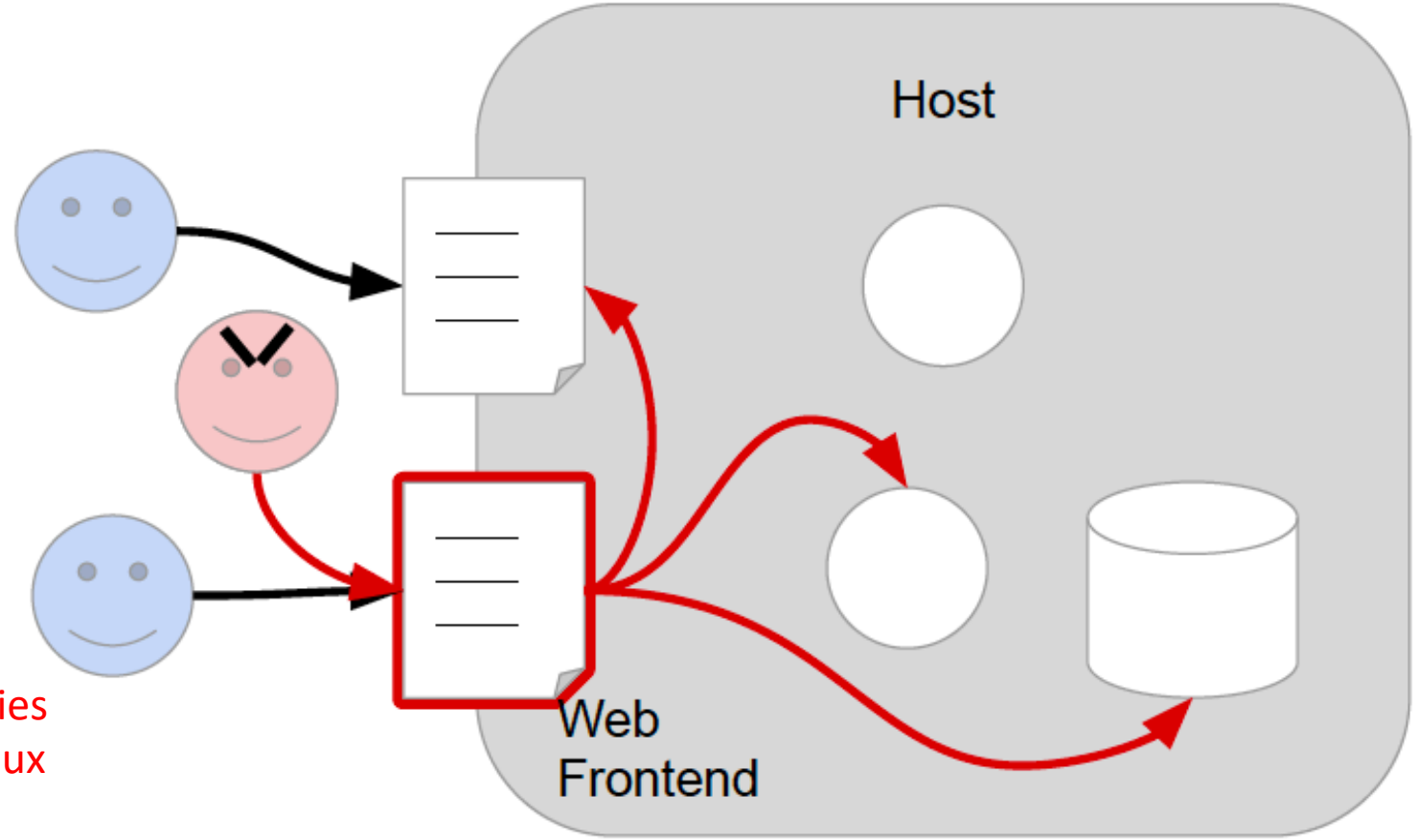
Kubernetes Dashboard



Attack Surface: Access to Host

1. Break out of the container using container or kernel exploits
2. Attack the Kubelet
3. Attack other containers running on the same host

Mitigate: Sandboxed Pods, Capabilities
Seccomp, Apparmor, SELinux



Security Context

```
apiVersion: v1
kind: Pod
metadata:
        name: security-context-demo
spec:
        securityContext:
                runAsUser: 1000
                readOnlyRootFilesystem: true
                allowPrivilegeEscalation: false
```

Seccomp, AppArmor, SELinux

```
metadata:
    annotations:
        seccomp.security.alpha.kubernetes.io/pod: runtime/default
        container.apparmor.security.beta.kubernetes.io/hello: runtime/default
securityContext:
    seLinuxOptions:
        level: "s0:c123,c456"
```

Security Context (1)

Field	Description
`allowPrivilegeEscalation` *boolean*	AllowPrivilegeEscalation controls whether a process can gain more privileges than its parent process. This bool directly controls if the no_new_privs flag will be set on the container process. AllowPrivilegeEscalation is true always when the container is: 1) run as Privileged 2) has CAP_SYS_ADMIN
`capabilities` *Capabilities*	The capabilities to add/drop when running containers. Defaults to the default set of capabilities granted by the container runtime.
`privileged` *boolean*	Run container in privileged mode. Processes in privileged containers are essentially equivalent to root on the host. Defaults to false.
`procMount` *string*	procMount denotes the type of proc mount to use for the containers. The default is DefaultProcMount which uses the container runtime defaults for readonly paths and masked paths. This requires the ProcMountType feature flag to be enabled.
`readOnlyRootFilesystem` *boolean*	Whether this container has a read-only root filesystem. Default is false.

Security Context (2)

`runAsGroup` *integer*	The GID to run the entrypoint of the container process. Uses runtime default if unset. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.
`runAsNonRoot` *boolean*	Indicates that the container must run as a non-root user. If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does. If unset or false, no such validation will be performed. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.
`runAsUser` *integer*	The UID to run the entrypoint of the container process. Defaults to user specified in image metadata if unspecified. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.
`seLinuxOptions` *SELinuxOptions*	The SELinux context to be applied to the container. If unspecified, the container runtime will allocate a random SELinux context for each container. May also be set in PodSecurityContext. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.
`seccompProfile` *SeccompProfile*	The seccomp options to use by this container. If seccomp options are provided at both the pod & container level, the container options override the pod options.
`windowsOptions` *WindowsSecurityContextOptions*	The Windows specific settings applied to all containers. If unspecified, the options from the PodSecurityContext will be used. If set in both SecurityContext and PodSecurityContext, the value specified in SecurityContext takes precedence.

Capabilities

```
$ cat /proc/$$/task/$$/status
....
CapEff: 0000000000000000
CapPrm: 0000000000000000
CapInh: 0000000000000000
CapAmb: 0000000000000000
CapBnd: 0000001fffffffff
securebits
```

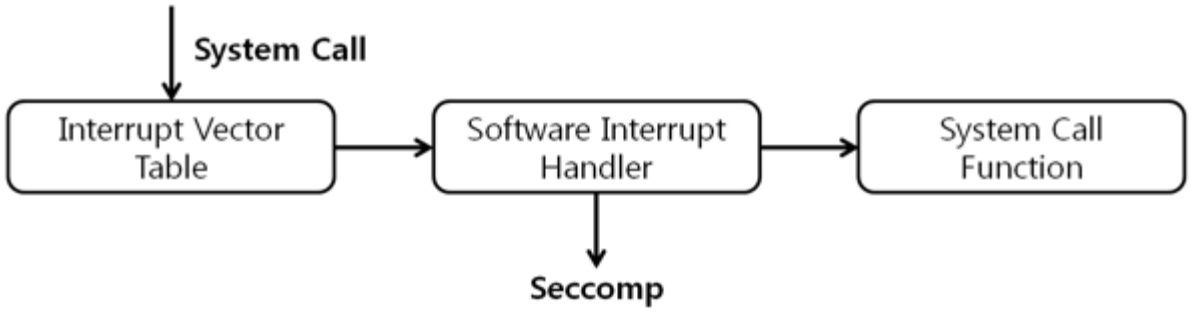
- Traditional UNIX privilege scheme
  - Process running with EUID 0 (superuser): all privilege actions allowed
  - Process running with EUID ≠ 0 : no privilege actions allowed
- Capabilities
  - Since kernel 2.2 (late 90's), 38 capabilities total
  - Collections of distinct privileges that can be enabled per process (thread)
  - CAP_SYS_BOOT, CAP_SYS_ADMIN, CAP_CHOWN, CAP_SYS_TIME, CAP_KILL …
  - capabilities(7), libcap(3)
  - 5 capability sets (effective, permitted, inheritable, ambient, bounding)
  - capset(2), capget(2), prctl
  - systemd: Default, User=…, AmbientCapabilities=X, CapabilityBoundingSet=X

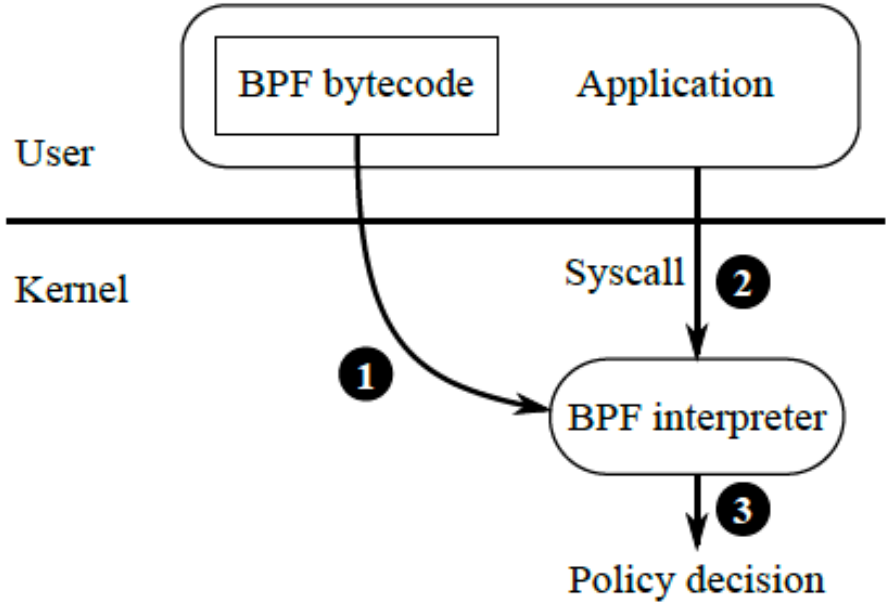
Dockerd Capabilities

- By default, Docker starts containers with a restricted set of capabilities
  - SETPCAP, MKNOD, AUDIT_WRITE, CHOWN, NET_RAW, DAC_OVERRIDE, FOWNER, FSETID, KILL, SETGID, SETUID, NET_BIND_SERVICE, SYS_CHROOT, SETFCAP
  - Removed: NET_ADMIN, SYS_ADMIN, …
- Activated command
  - Runs as native process, by default under EUID 0
  - Runs with selected set of capabilities
- Capability-related arguments
  - Add capabilities: --cap-add *list*
  - Drop capabilities: --cap-drop *list*
  - User: --user name|uid (no capabilities)

Seccomp

- Since kernel 2.6.12 (2005)
- Seccomp-bpf
  - Allows filtering of system calls using a configurable policy implemented using Berkeley Packet Filter rules
  - For Google Chrome



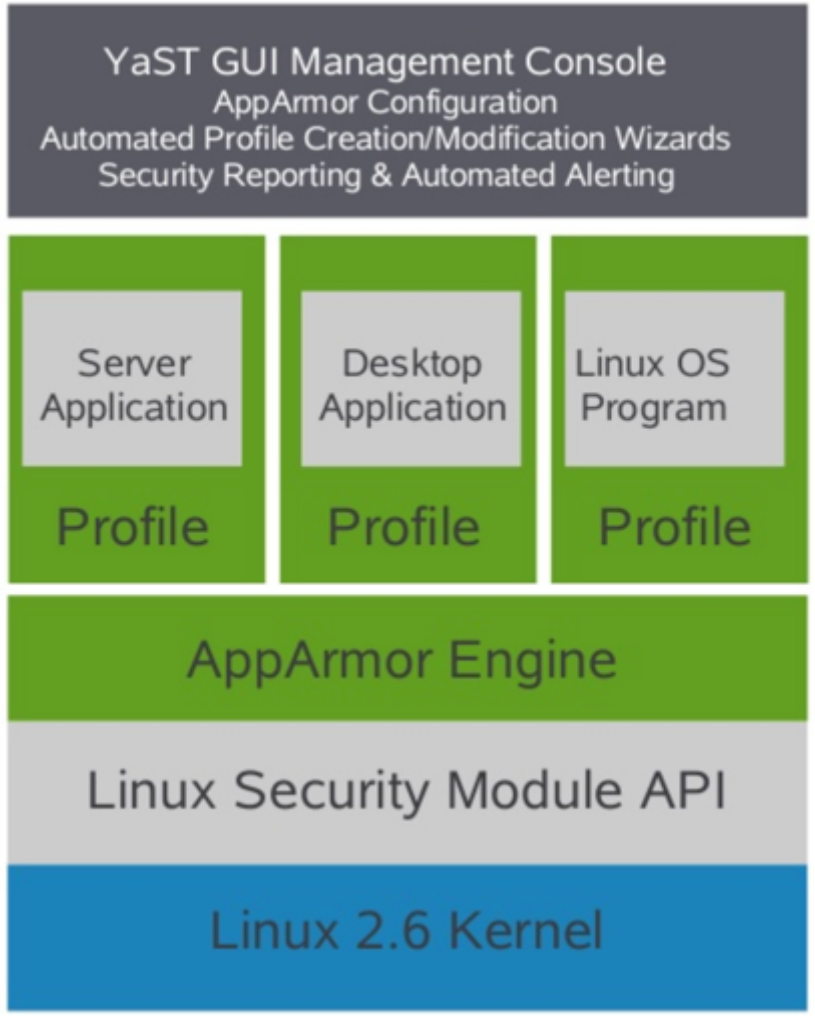


Mandatory Access Control Systems: SELinux

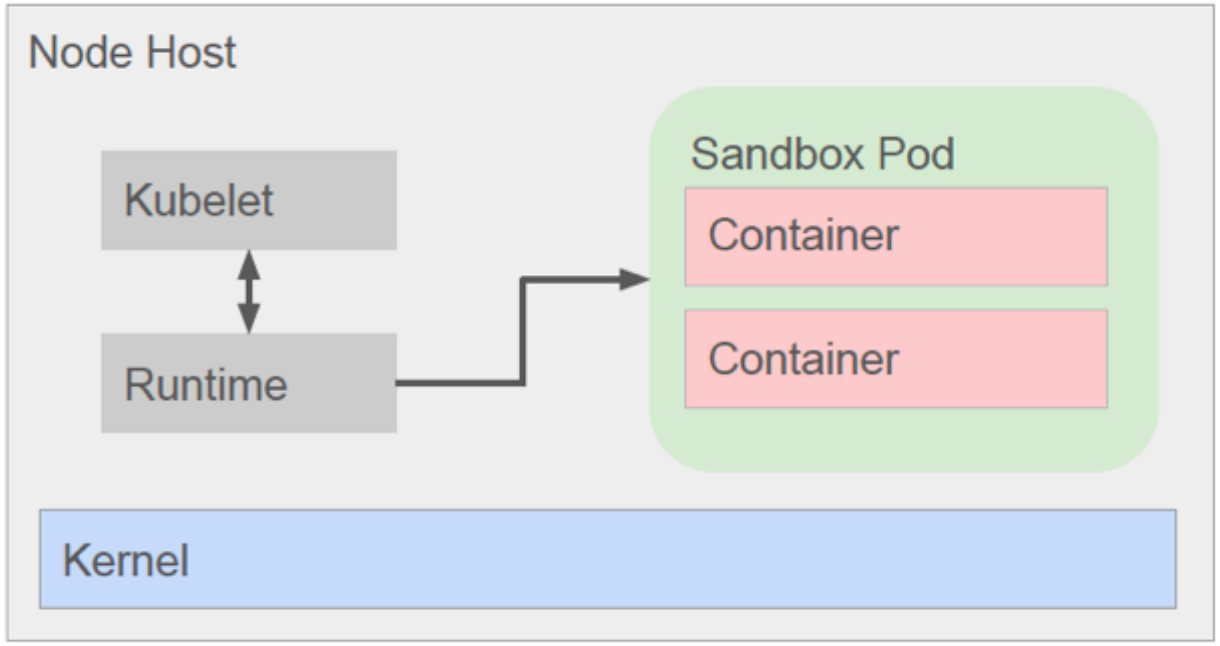
- SELinux (Security Enhanced Linux)
  - sudo setenforce 1 → sudo sestatus
  - Set security labels, 3 modes (enforce, permissive, disable)
  - Even if Apache HTTP Server is compromised, an attacker cannot use that process to read files in user home directories by default, unless a specific SELinux policy rule was added or configured to allow such access
- Protecting host file system from container breakout

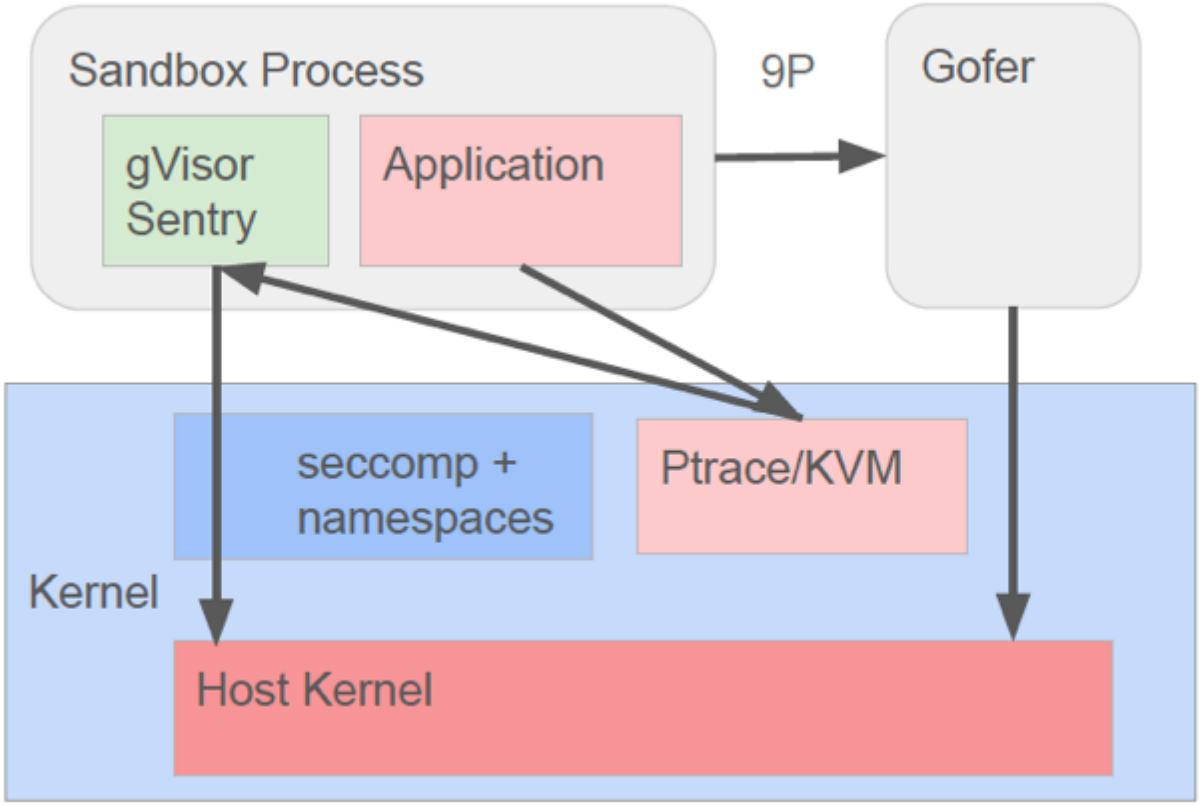
Mandatory Access Control Systems: AppArmor

- AppArmor
  - Restrict programs' capabilities with per-program profiles
  - Profiles: /etc/apparmor.d
  - sudo apparmor_parser /etc/apparmor.d/… → sudo aa-status
  - 2 modes (enforcement, complain)



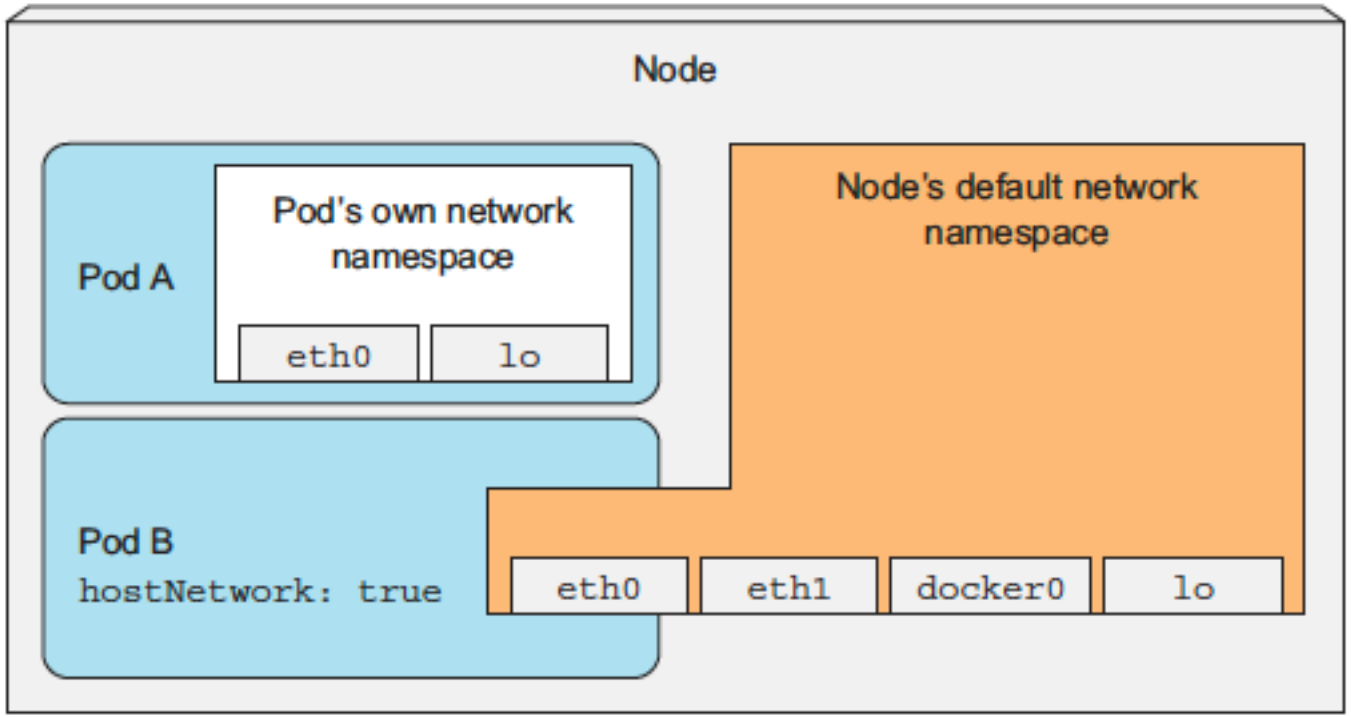
Sandboxing


Node Host
Kubelet
Runtime
Sandbox Pod
Container
Container
Kernel

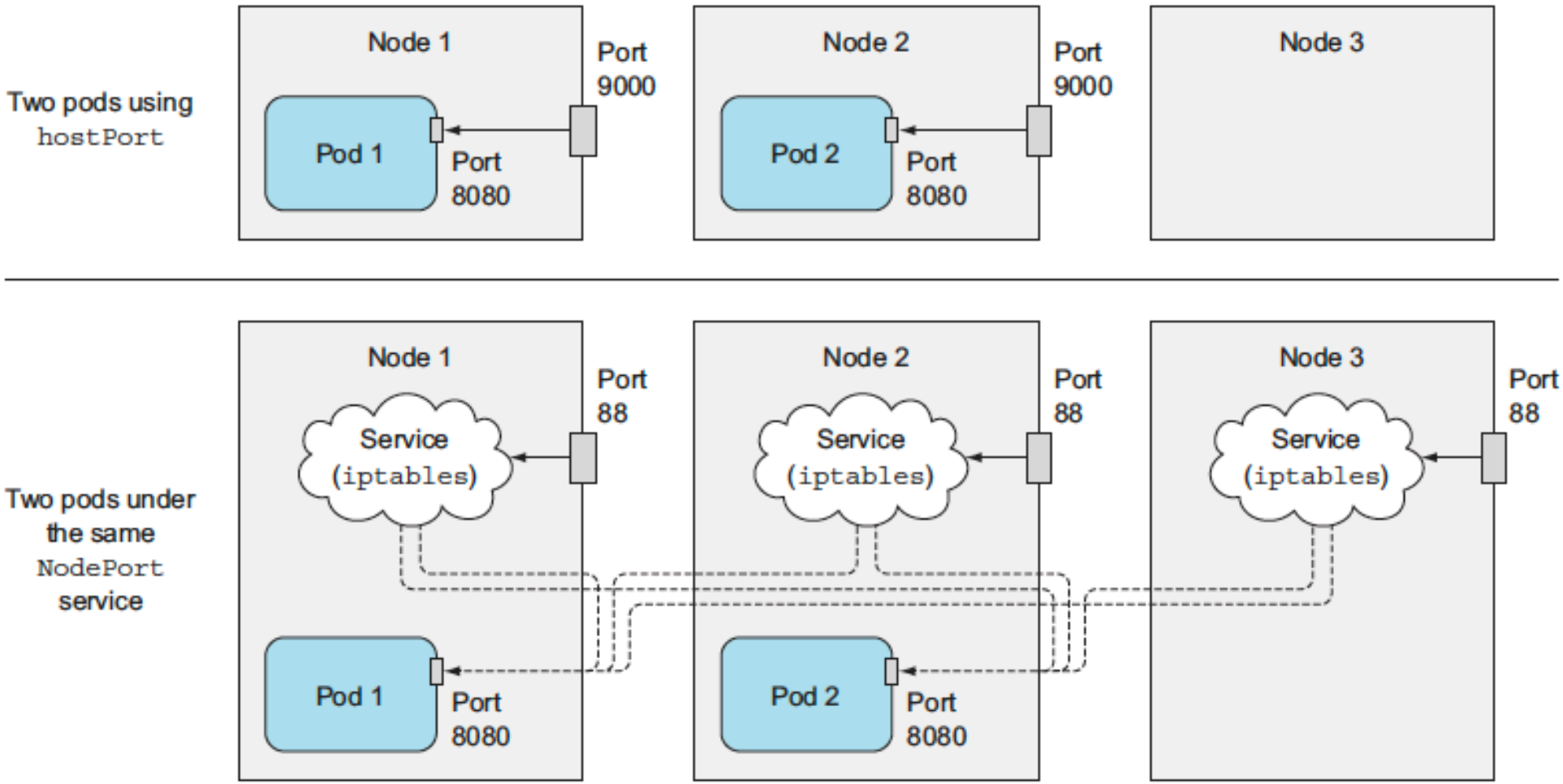

Sandbox Process
gVisor
Sentry
Application
9P
Gofer
seccomp +
namespaces
Ptrace/KVM
Kernel
Host Kernel

hostNetwork

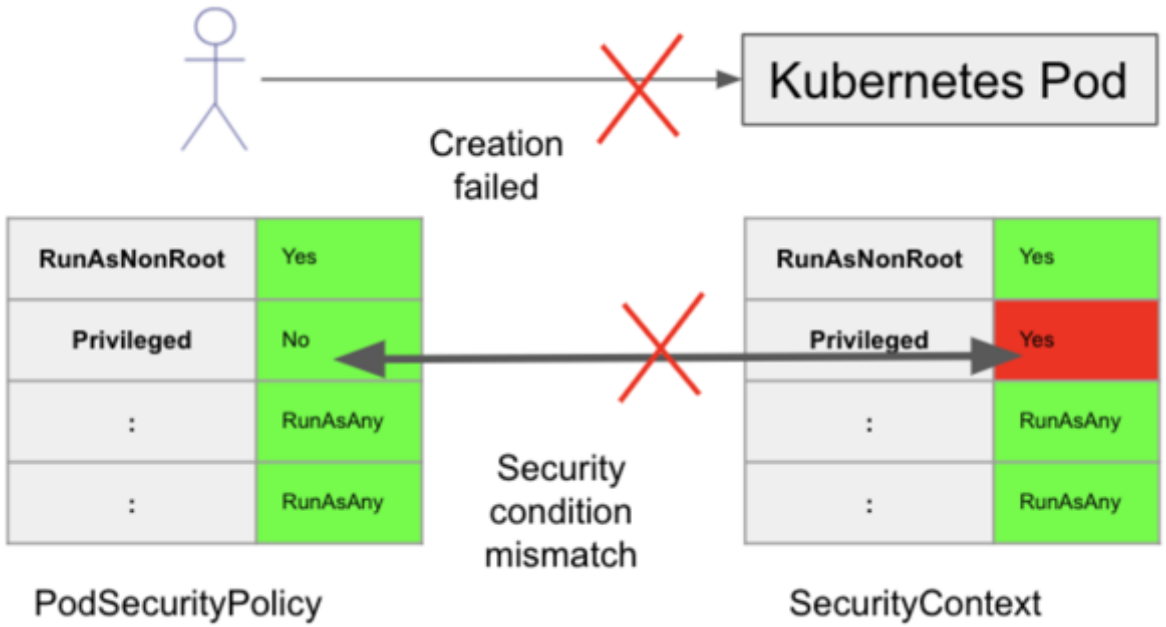
```
spec:
  hostNetwork: true
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep"
```

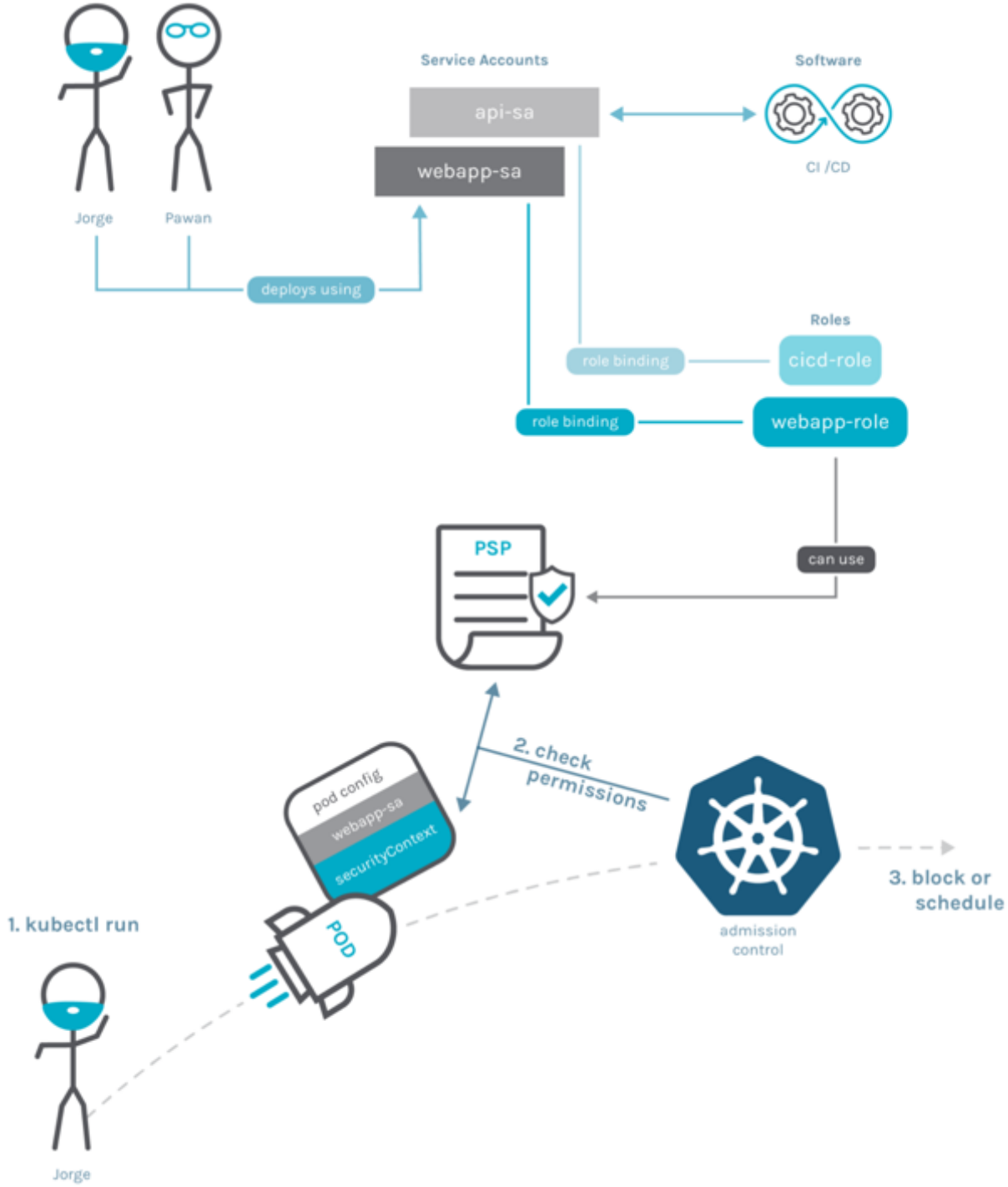


hostPort

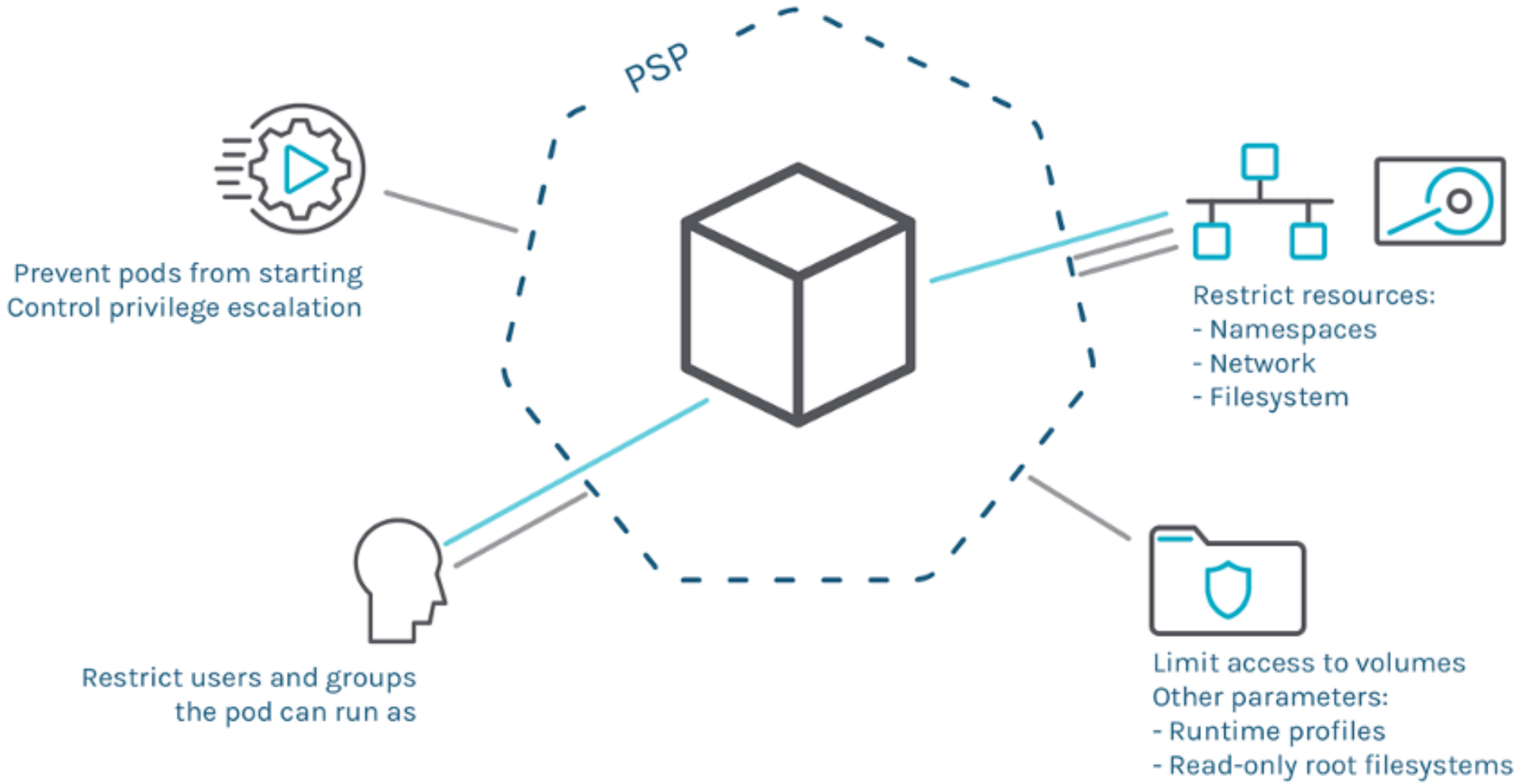

Two pods using hostPort
Node 1
Port 9000
Pod 1
Port 8080
Node 2
Port 9000
Pod 2
Port 8080
Node 3
Two pods under the same NodePort service
Node 1
Port 88
Service (iptables)
Pod 1
Port 8080
Node 2
Port 88
Service (iptables)
Pod 2
Port 8080
Node 3
Port 88
Service (iptables)

Pod Security Policy





Pod Security Policy Benefits



PSP Examples

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: default
spec:
  hostIPC: false
  hostPID: false
  hostNetwork: false
  hostPorts:
  - min: 10000
    max: 11000
  - min: 13000
    max: 14000
  privileged: false
  readOnlyRootFilesystem: true
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  volumes:
  - '*'
```

Containers aren't allowed to use the host's IPC, PID, or network namespace.

They can only bind to host ports 10000 to 11000 (inclusive) or host ports 13000 to 14000.

Containers cannot run in privileged mode.

Containers can run as any user and any group.

They can also use any SELinux groups they want.

All volume types can be used in pods.

Containers are forced to run with a read-only root filesystem.

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
spec:
  allowedCapabilities:
  - SYS_TIME
  defaultAddCapabilities:
  - CHOWN
  requiredDropCapabilities:
  - SYS_ADMIN
  - SYS_MODULE
  ...
```

Allow containers to add the SYS_TIME capability.

Automatically add the CHOWN capability to every container.

Require containers to drop the SYS_ADMIN and SYS_MODULE capabilities.

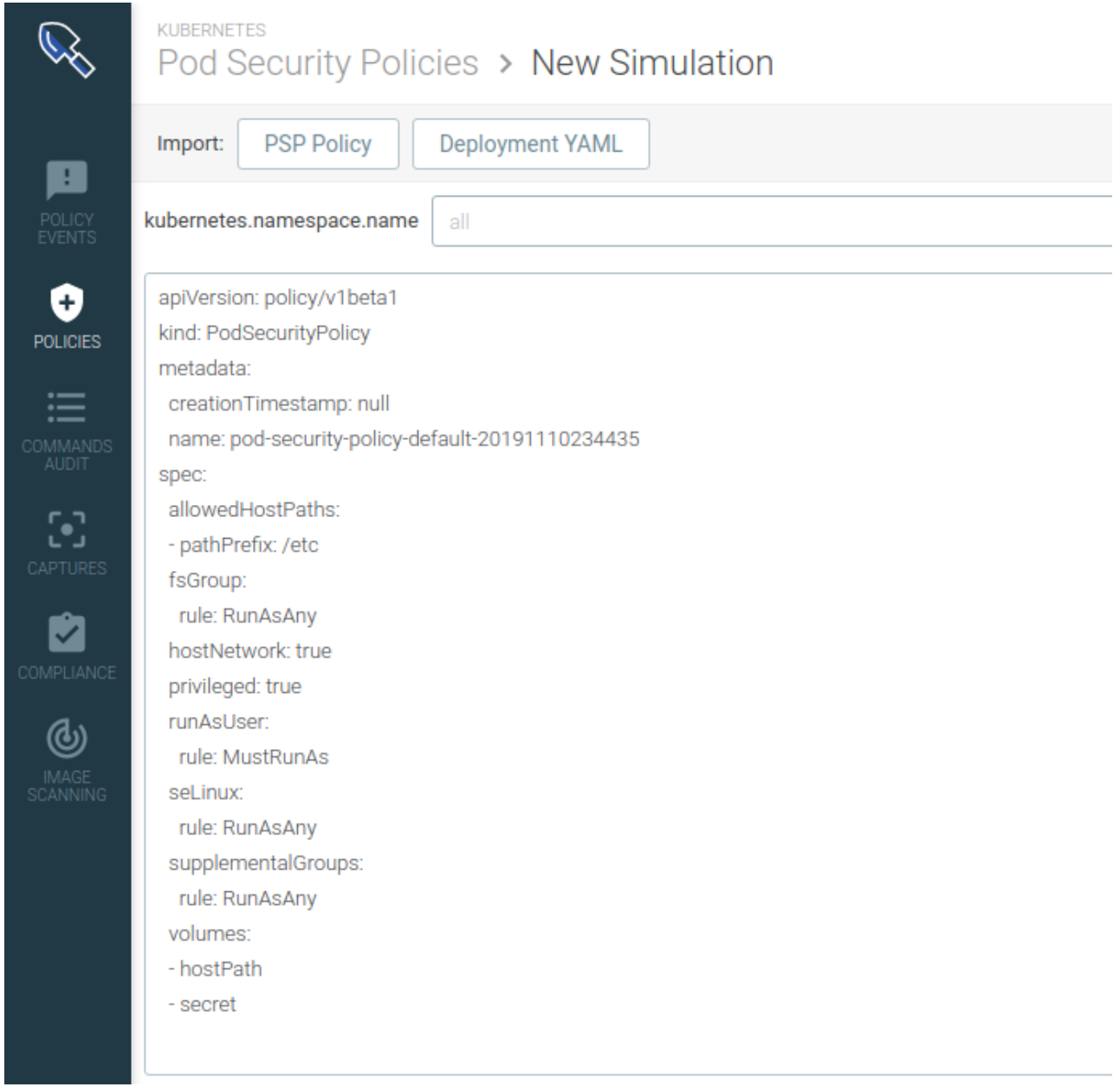
PSP Controls (1)

제어 측면	필드 이름
특권을 가진(privileged) 컨테이너의 실행	`privileged`
호스트 네임스페이스의 사용	`hostPID`, `hostIPC`
호스트 네트워킹과 포트의 사용	`hostNetwork`, `hostPorts`
볼륨 유형의 사용	`volumes`
호스트 파일시스템의 사용	`allowedHostPaths`
특정 FlexVolume 드라이버의 허용	`allowedFlexVolumes`
파드 볼륨을 소유한 FSGroup 할당	`fsGroup`
읽기 전용 루트 파일시스템 사용 필요	`readOnlyRootFilesystem`

PSP Controls (2)

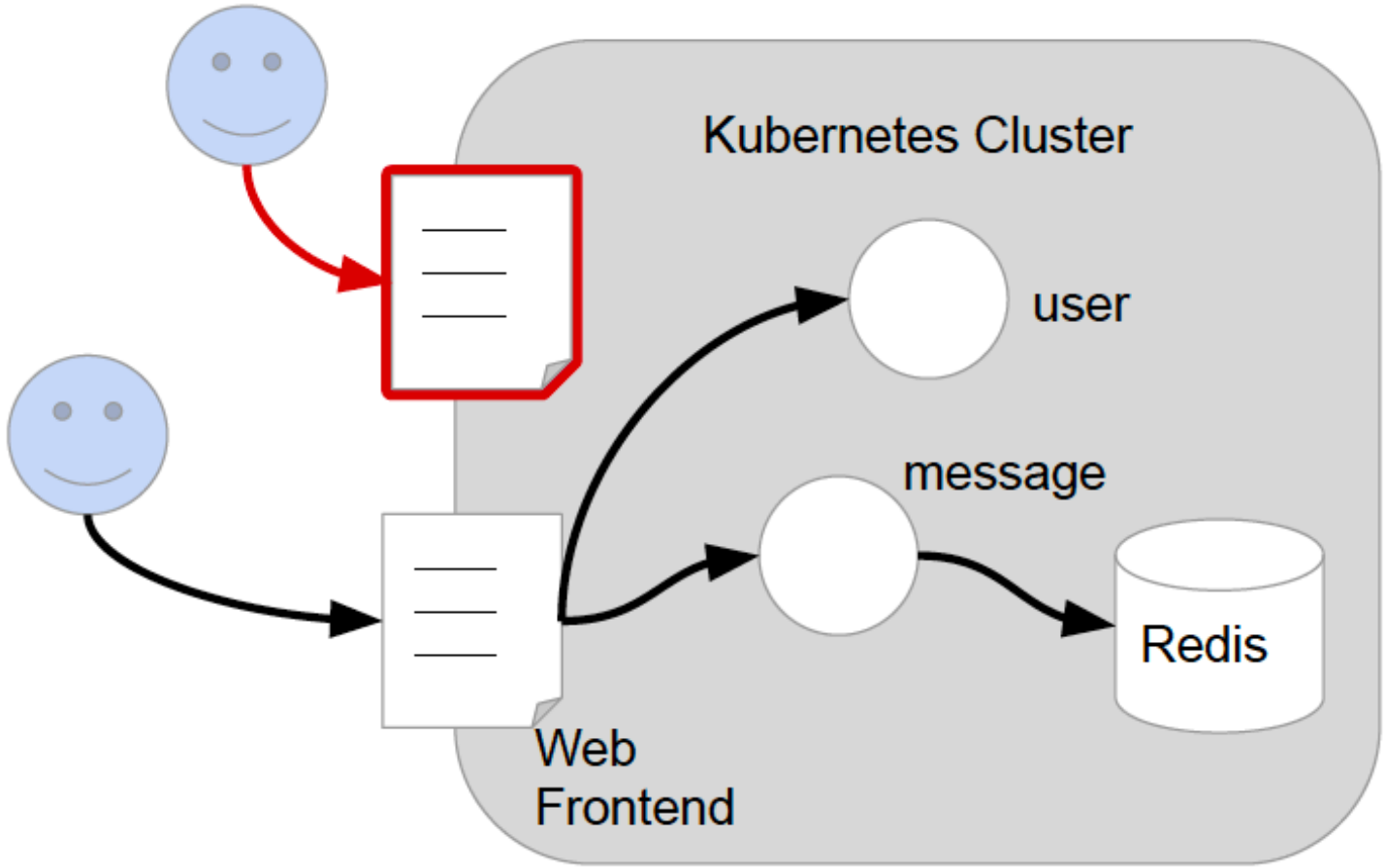
컨테이너의 사용자 및 그룹 ID	`runAsUser`, `runAsGroup`, `supplementalGroups`
루트 특권으로의 에스컬레이션 제한	`allowPrivilegeEscalation`, `defaultAllowPrivilegeEscalation`
리눅스 기능	`defaultAddCapabilities`, `requiredDropCapabilities`, `allowedCapabilities`
컨테이너의 SELinux 컨텍스트	`seLinux`
컨테이너에 허용된 Proc 마운트 유형	`allowedProcMountTypes`
컨테이너가 사용하는 AppArmor 프로파일	어노테이션
컨테이너가 사용하는 seccomp 프로파일	어노테이션
컨테이너가 사용하는 sysctl 프로파일	`forbiddenSysctls`, `allowedUnsafeSysctls`

Sysdig Secure

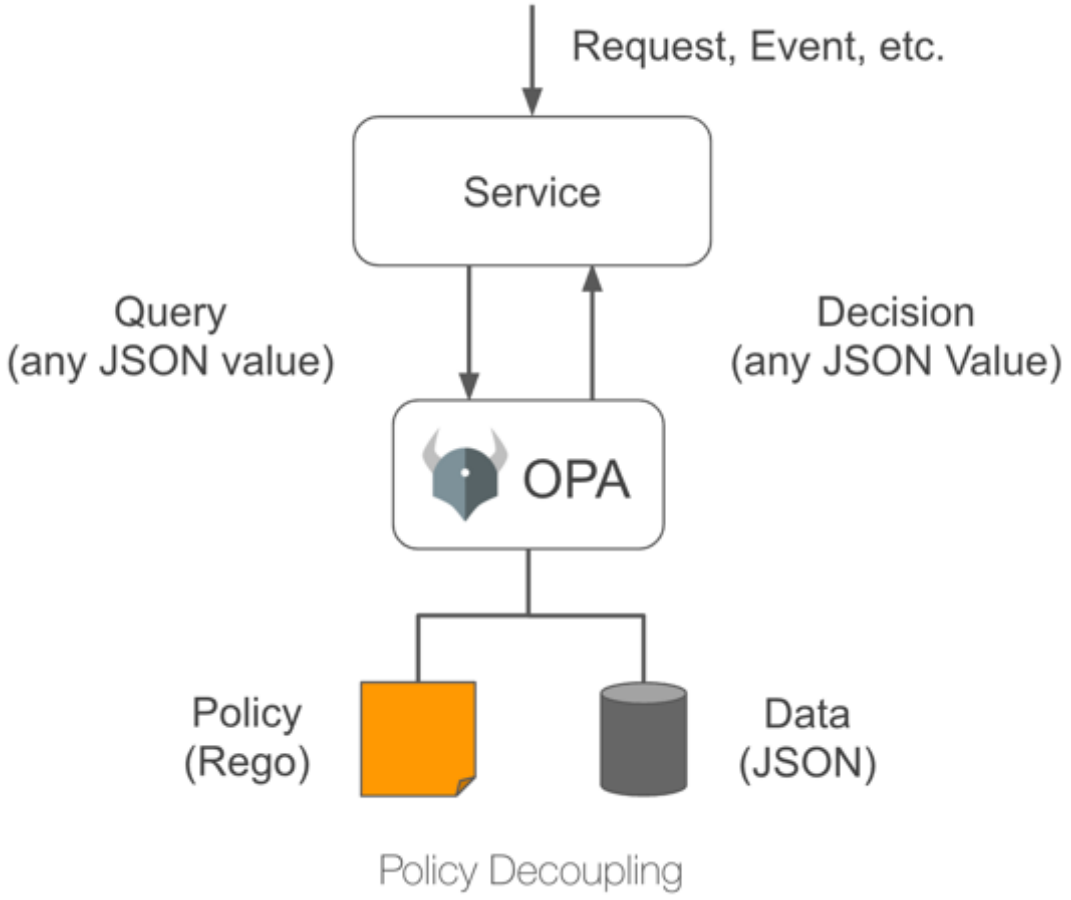


Attack Surface: Unsecured Pod

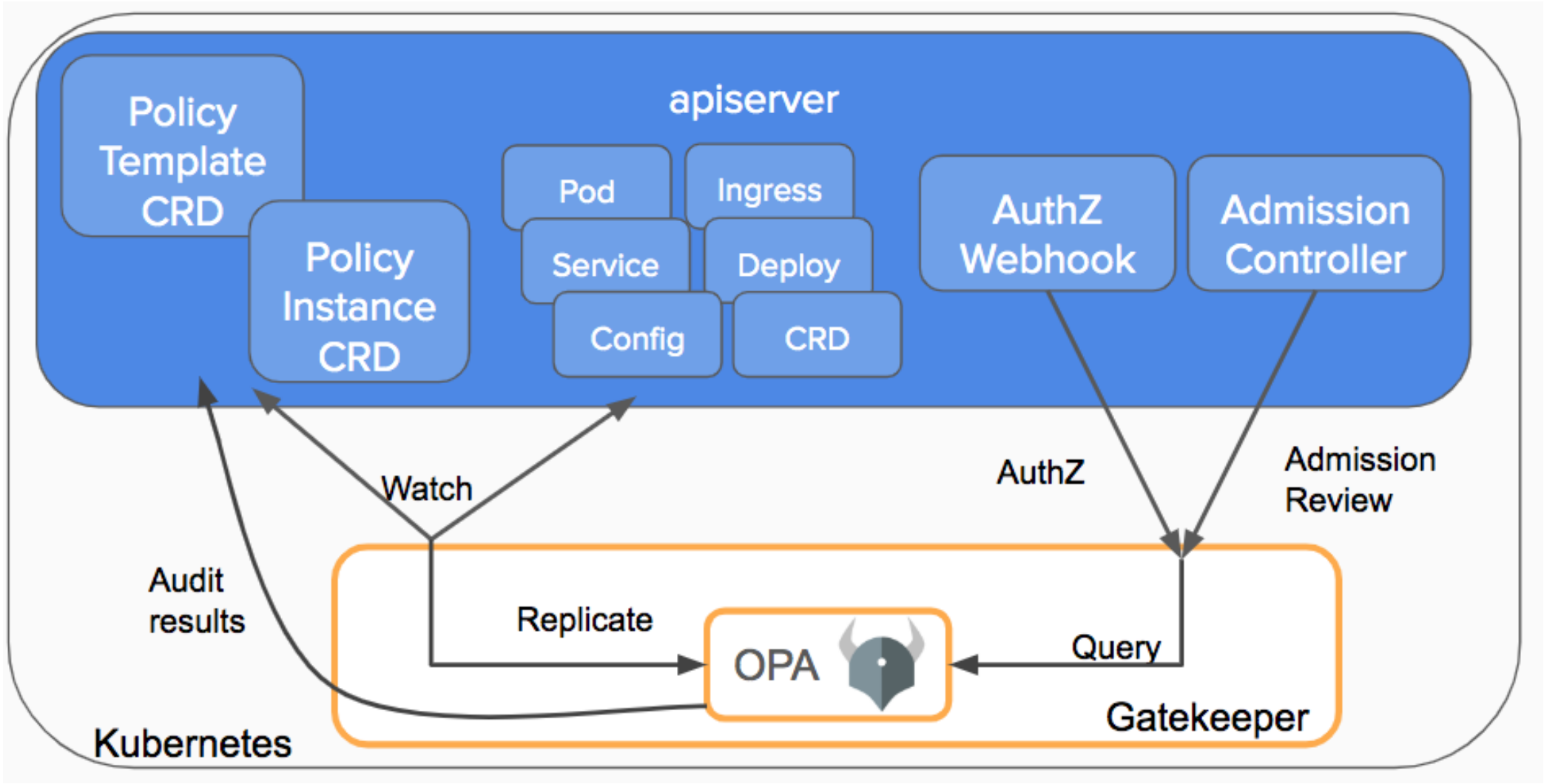
Mitigate: Open Security Policy

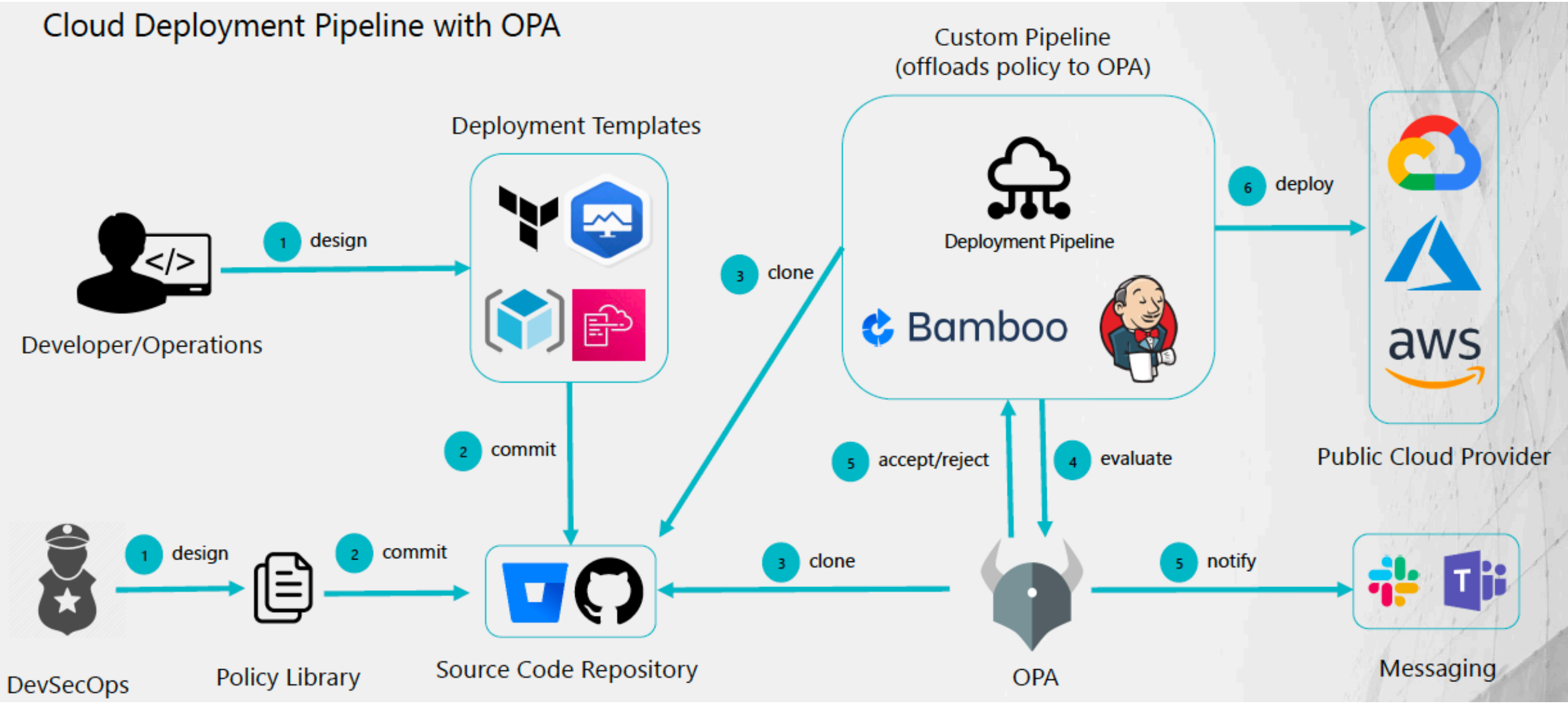


OPA (Open Policy Agent)

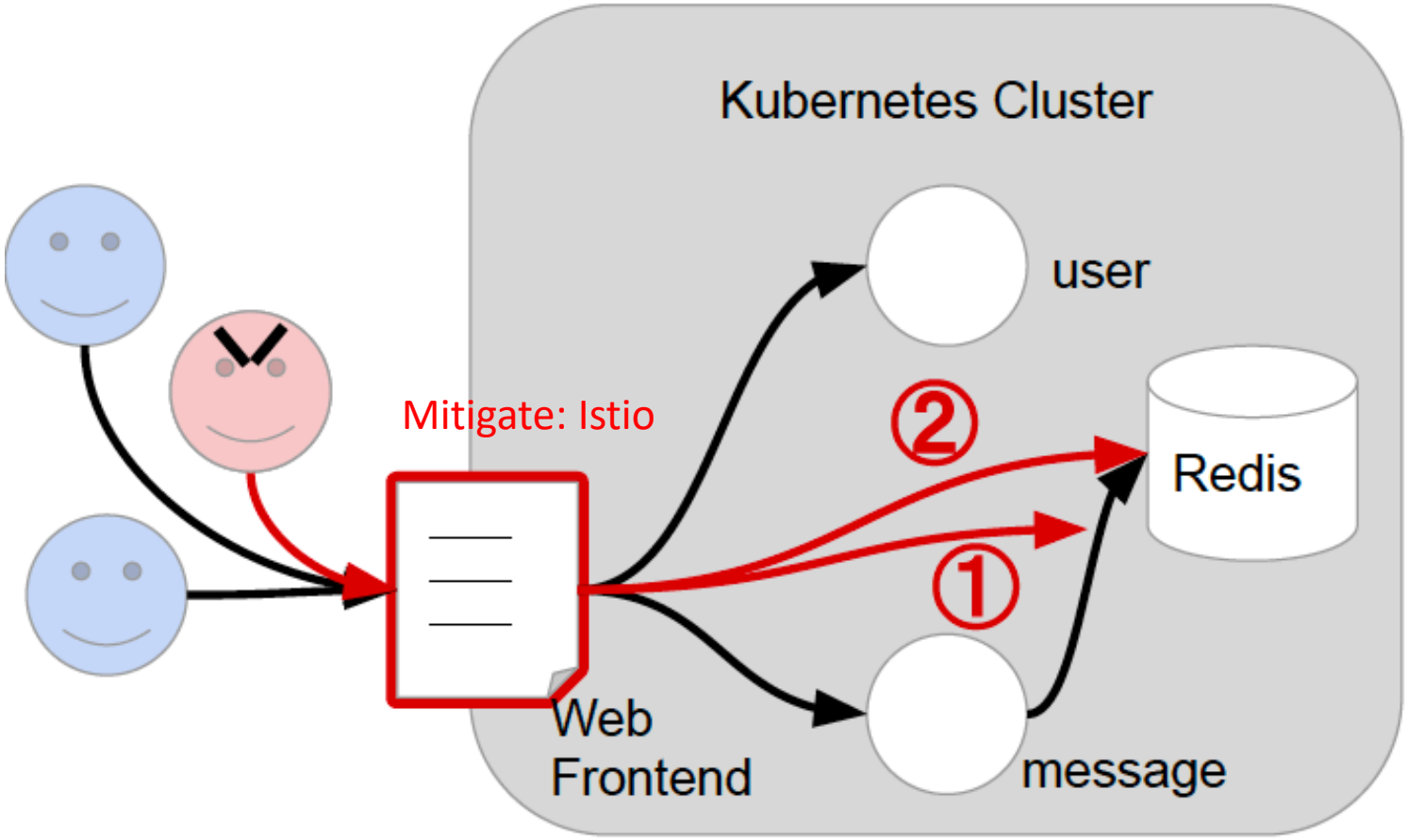


OPA Gatekeeper

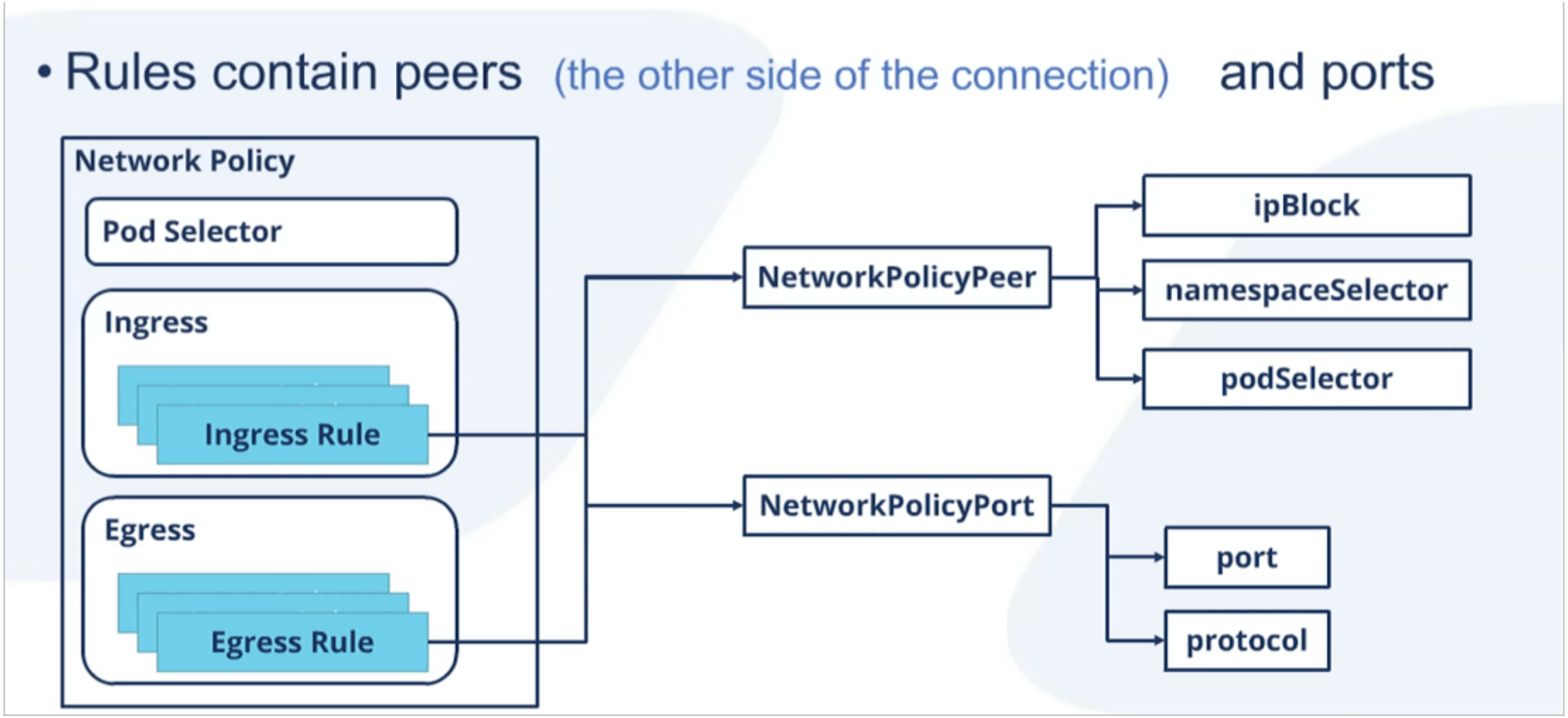



Cloud Deployment Pipeline with OPA
Custom Pipeline
(offloads policy to OPA)
Deployment Templates
Deployment Pipeline
Bamboo
Developer/Operations
1 design
2 commit
3 clone
4 evaluate
5 accept/reject
6 deploy
Public Cloud Provider
aws
DevSecOps
1 design
Policy Library
2 commit
Source Code Repository
3 clone
OPA
5 notify
Messaging

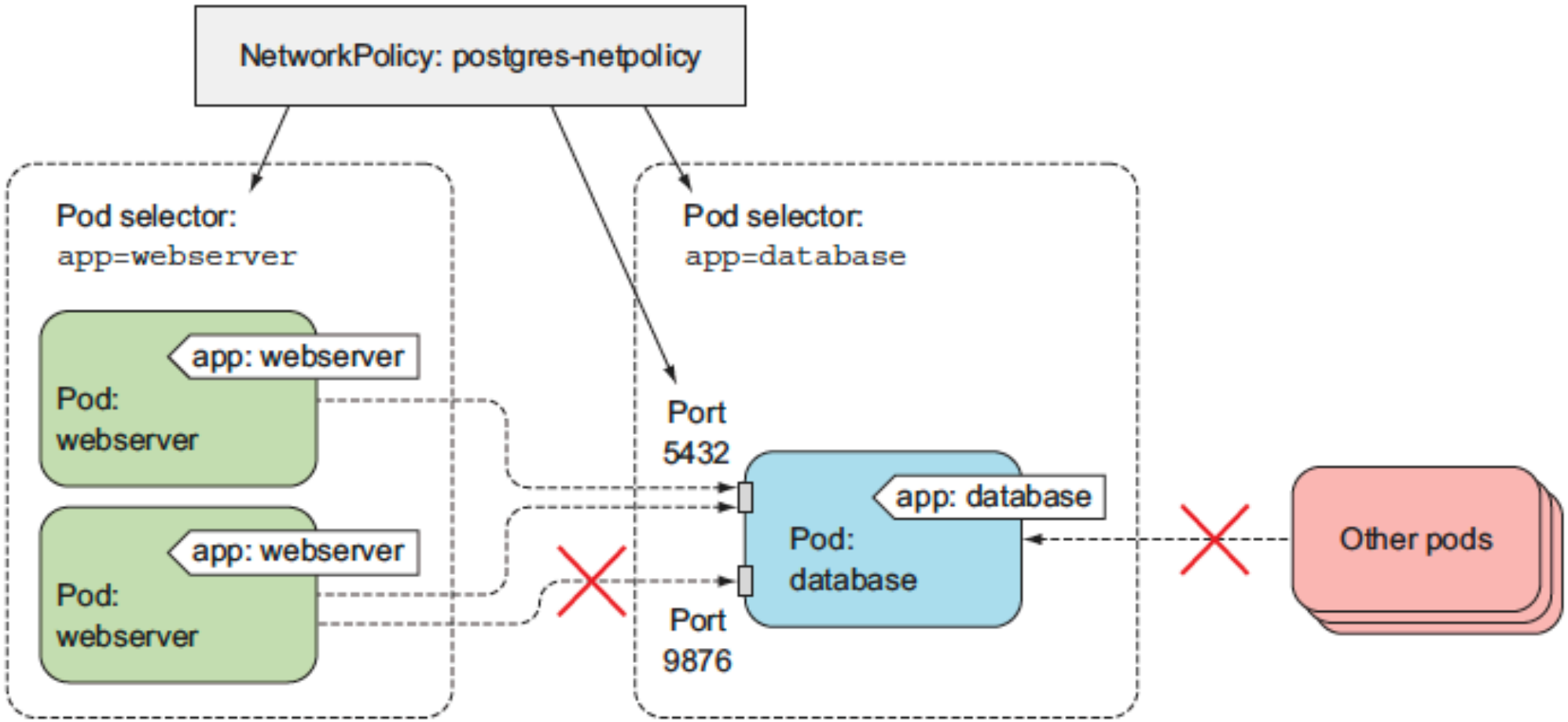
Attack Surface: Traffic Sniffing



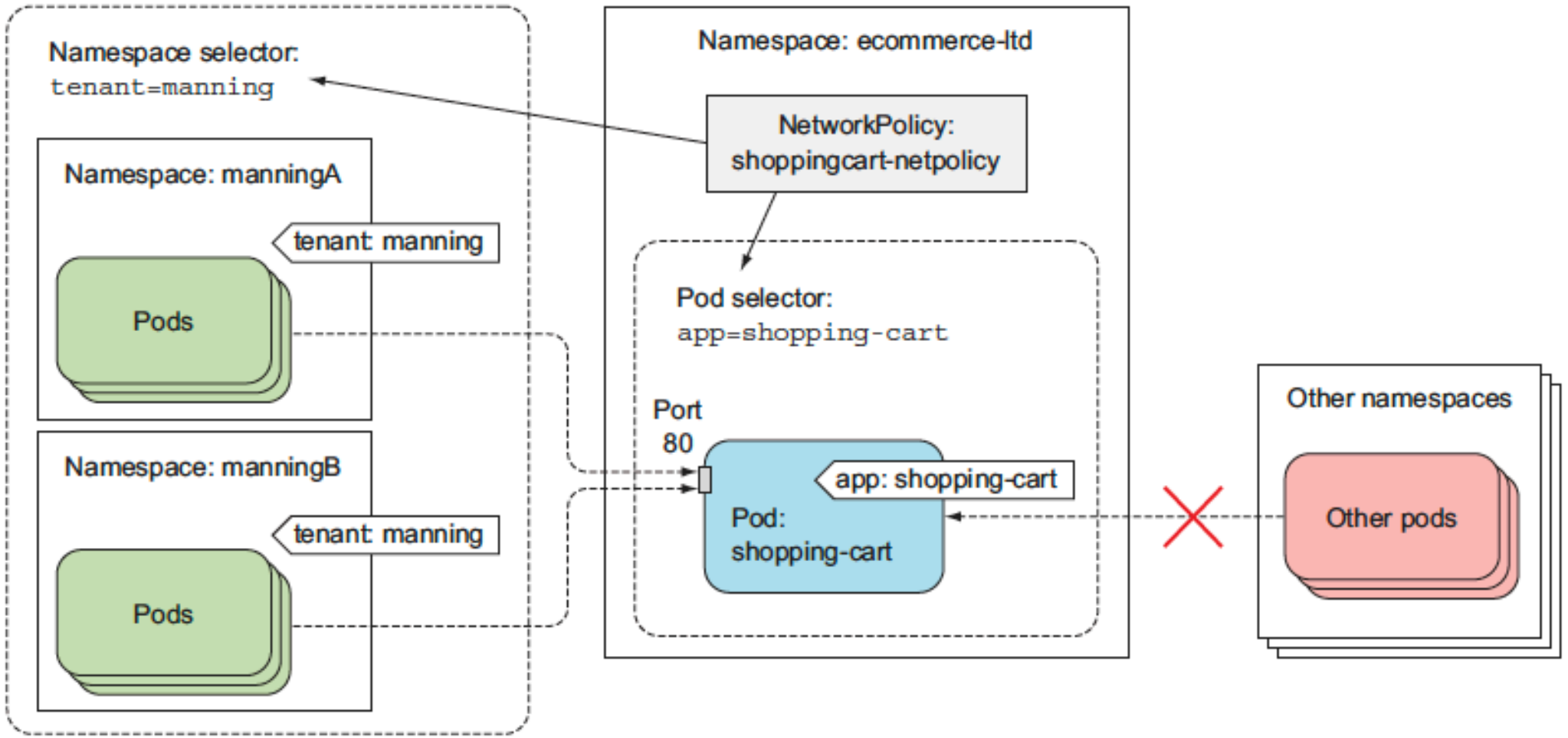
Network Policy



Network Policy: Pod Selector

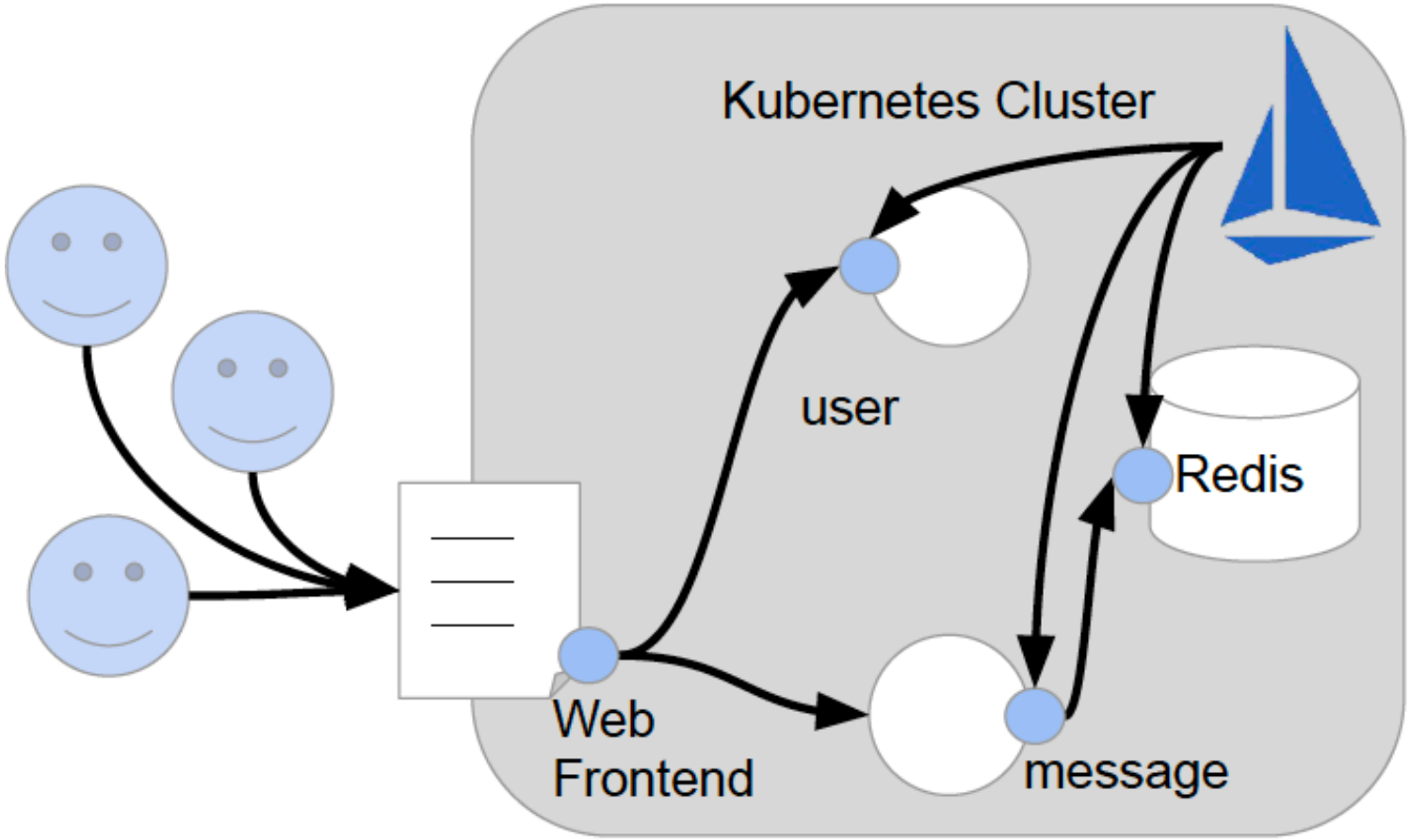


Network Policy: Namespace Selector

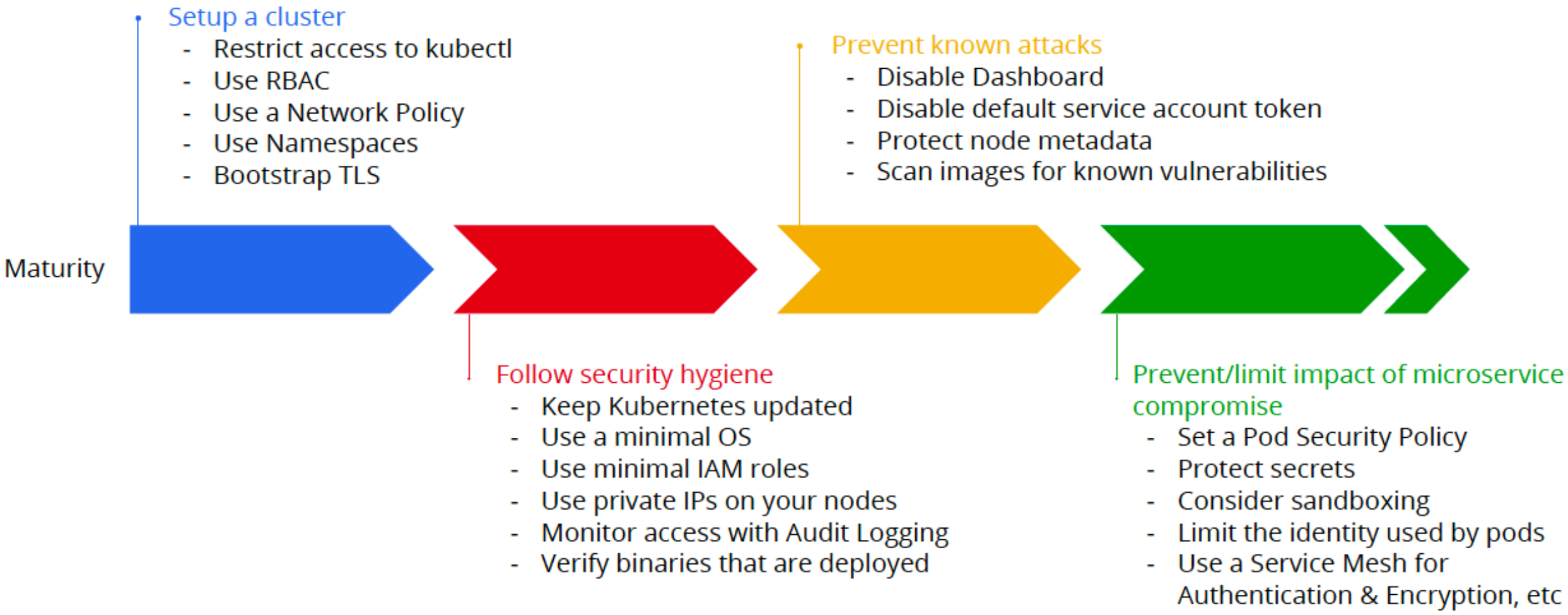


Istio

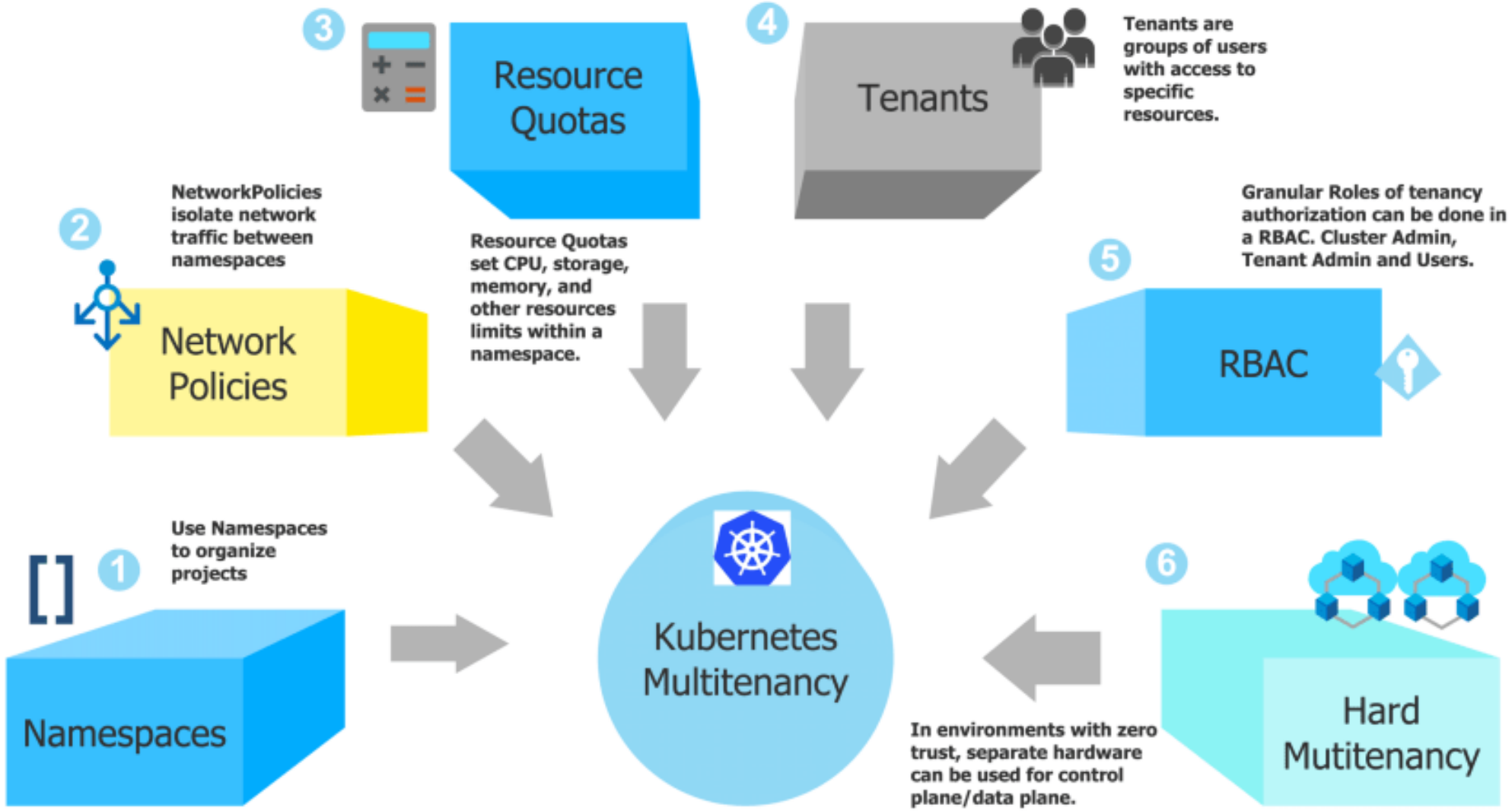
1. Proxy data between services
2. End-to-end encryption
3. Rolling certificates
4. Policy managed by central server



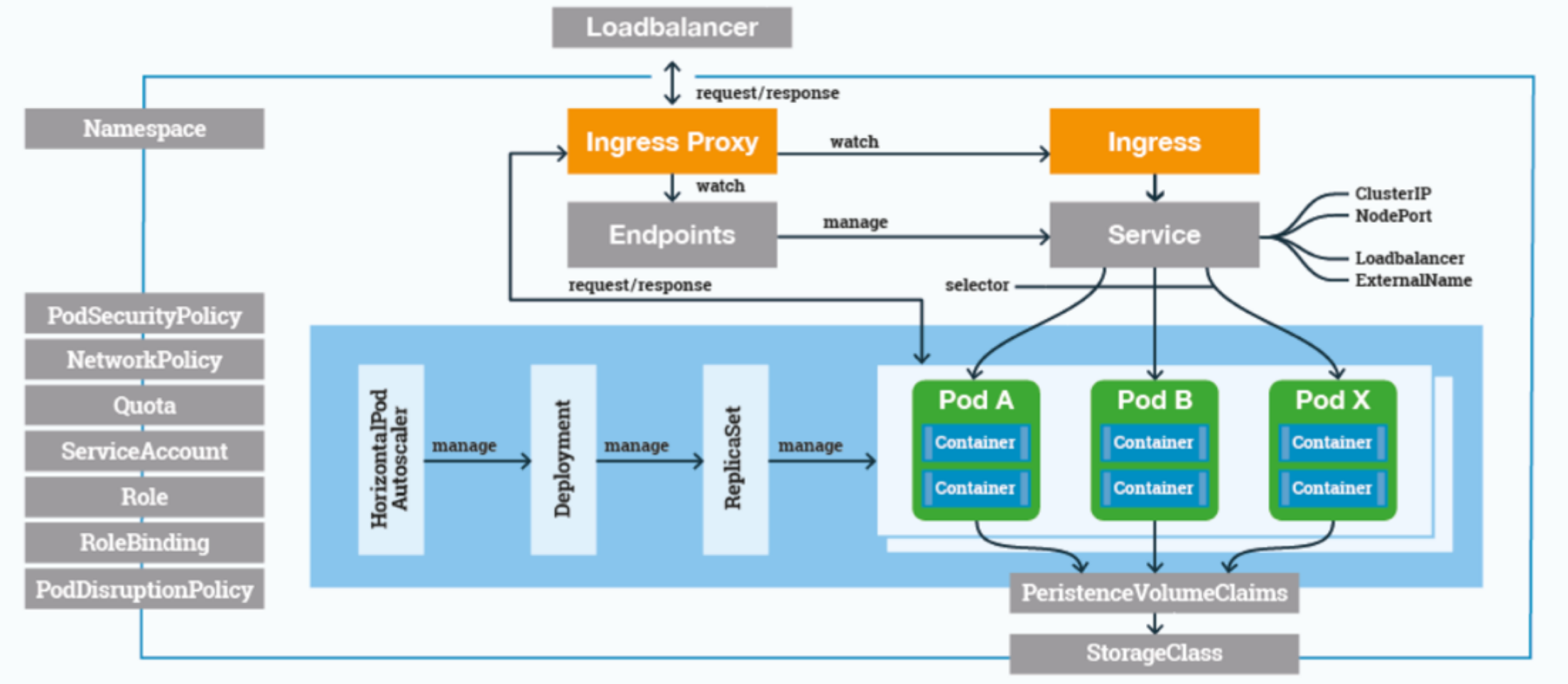
Kubernetes Security Journey

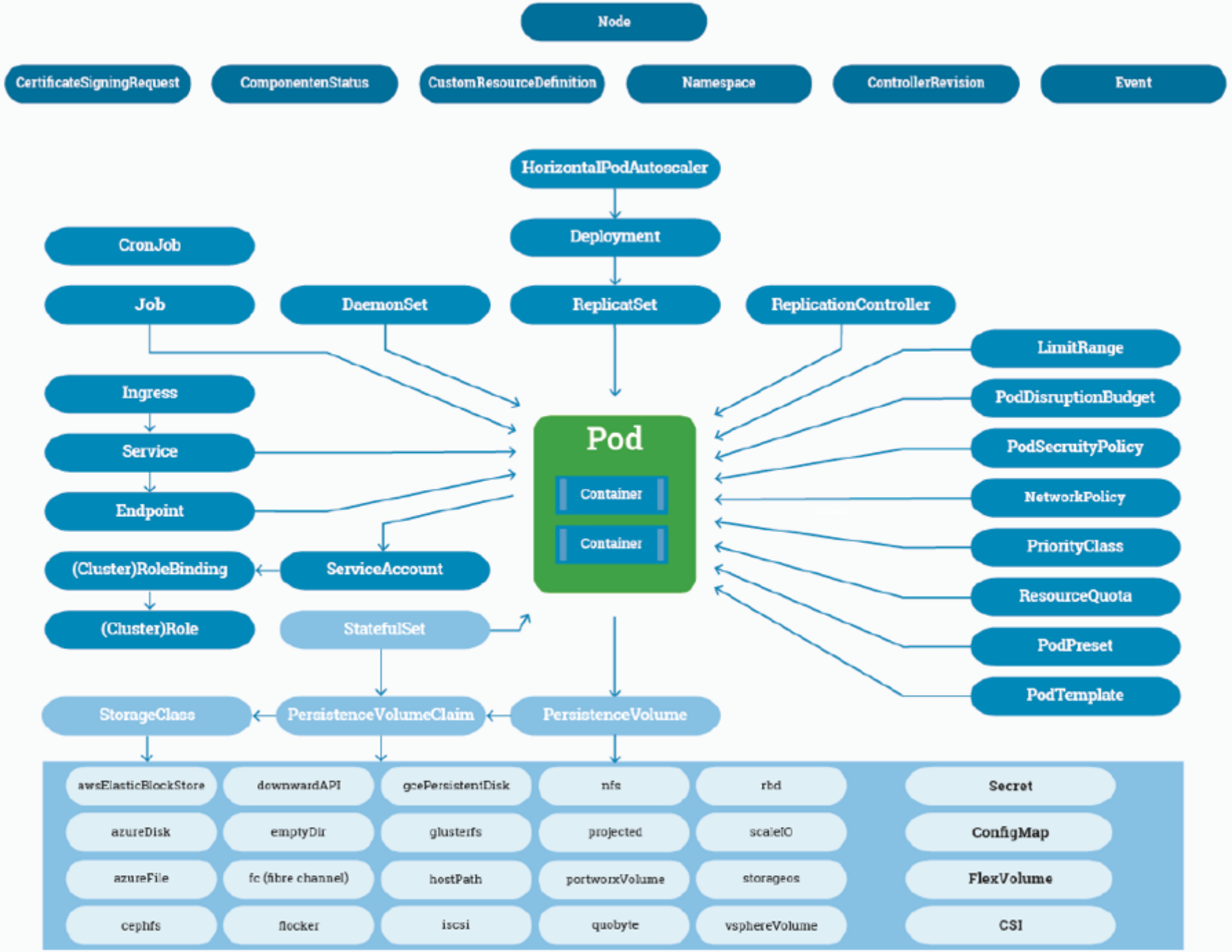


Kubernetes Multi-Tenancy Challenge

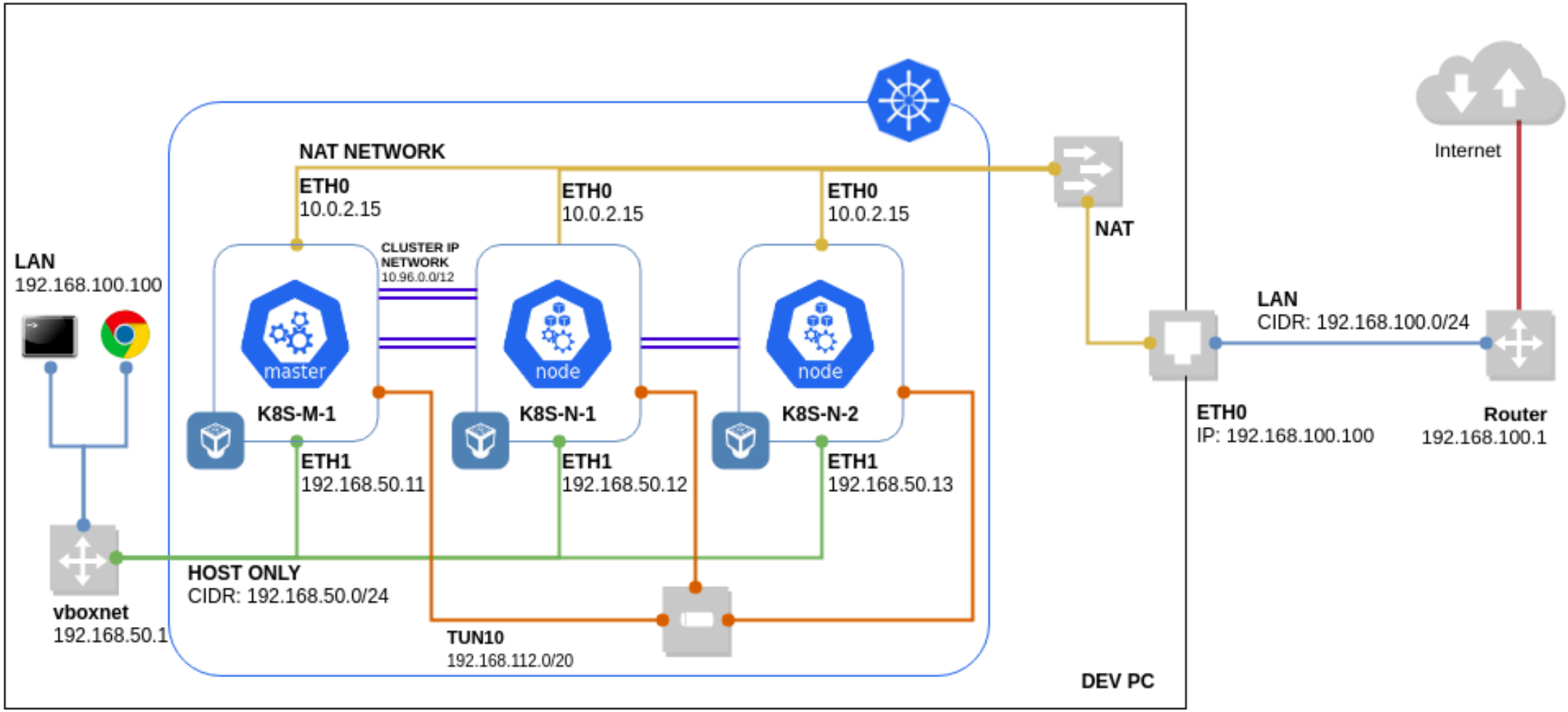


Kubernetes Service Architecture

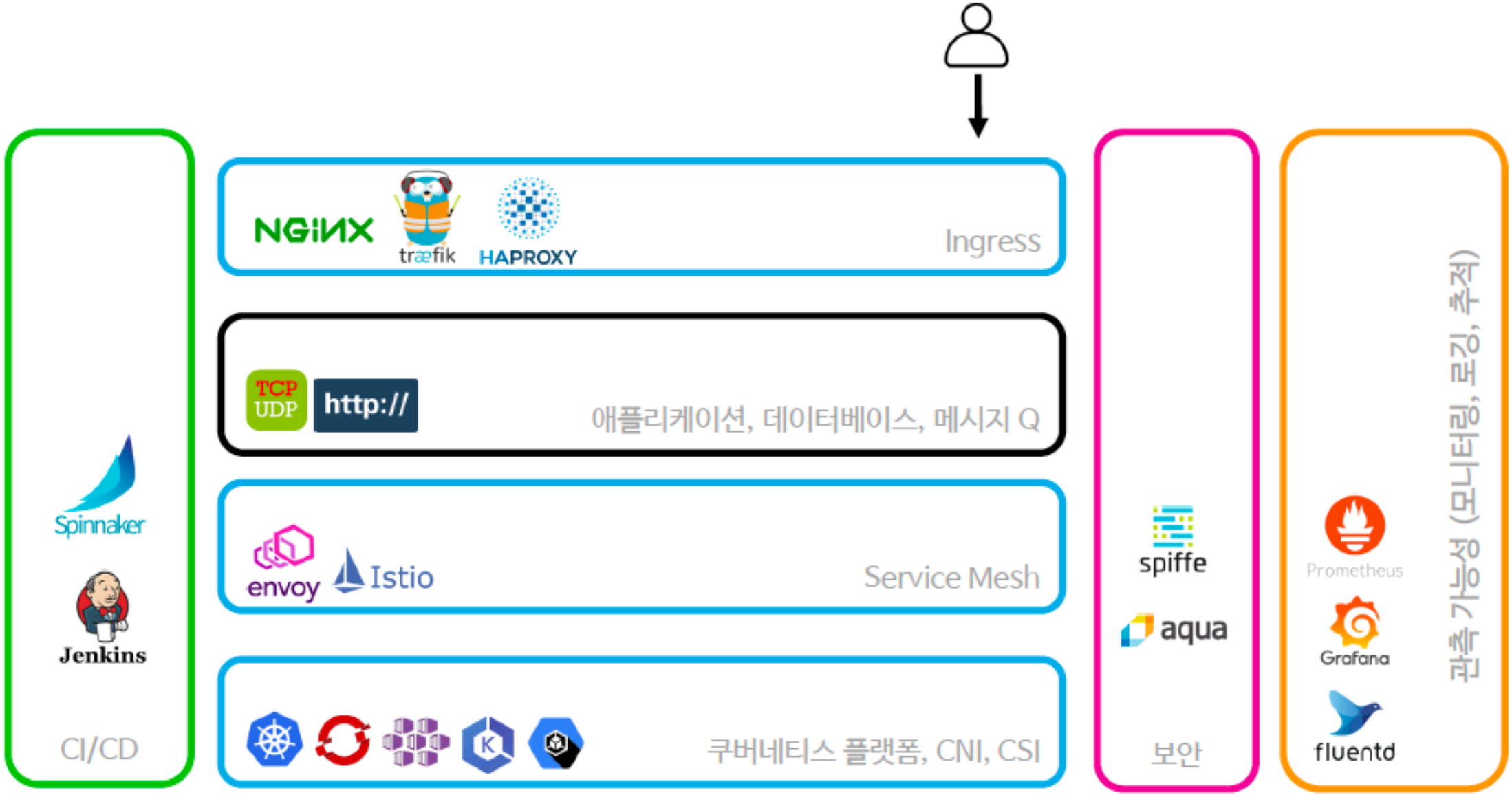



Node
CertificateSigningRequest
ComponentenStatus
CustomResourceDefinition
Namespace
ControllerRevision
Event
HorizontalPodAutoscaler
Deployment
CronJob
Job
DaemonSet
ReplicatSet
ReplicationController
LimitRange
Ingress
PodDisruptionBudget
Service
PodSecruityPolicy
Pod
Container
Endpoint
NetworkPolicy
Container
(Cluster)RoleBinding
ServiceAccount
PriorityClass
(Cluster)Role
StatefulSet
ResourceQuota
PodPreset
StorageClass
PersistenceVolumeClaim
PersistenceVolume
PodTemplate
awsElasticBlockStore
downwardAPI
gcePersistentDisk
nfs
rbd
Secret
azureDisk
emptyDir
glusterfs
projected
scaleIO
ConfigMap
azureFile
fc (fibre channel)
hostPath
portworxVolume
storageos
FlexVolume
cephfs
flocker
iscsi
quobyte
vsphereVolume
CSI

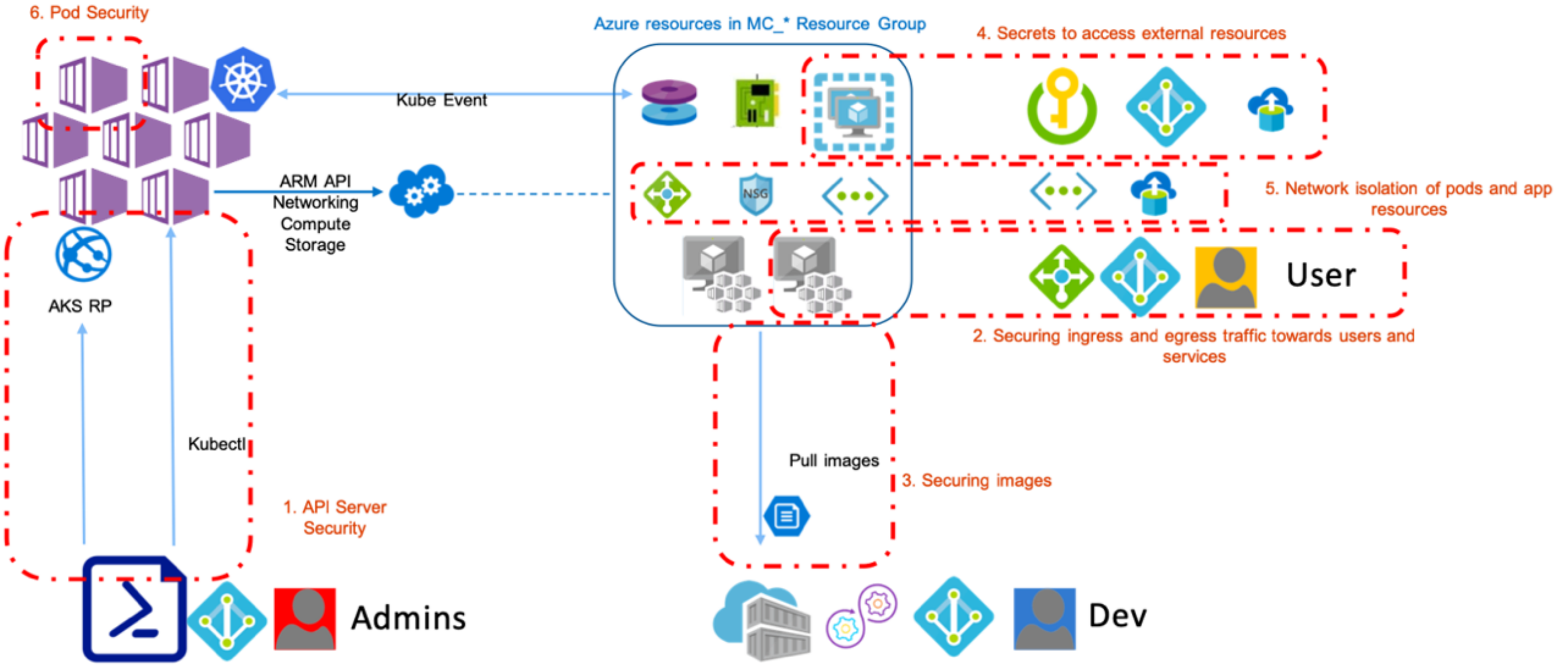
Local Kubernetes Cluster with VirtualBox



https://www.itwonderlab.com/en/ansible-kubernetes-vagrant-tutorial/



이해 관계자: DevOps | 개발자 | 플랫폼 관리자 | DevSecOps | SRE


6. Pod Security
Azure resources in MC_* Resource Group
4. Secrets to access external resources
Kube Event
ARM API
Networking
Compute
Storage
NSG
5. Network isolation of pods and app resources
User
2. Securing ingress and egress traffic towards users and services
AKS RP
Kubectl
Pull images
3. Securing images
1. API Server Security
Admins
Dev