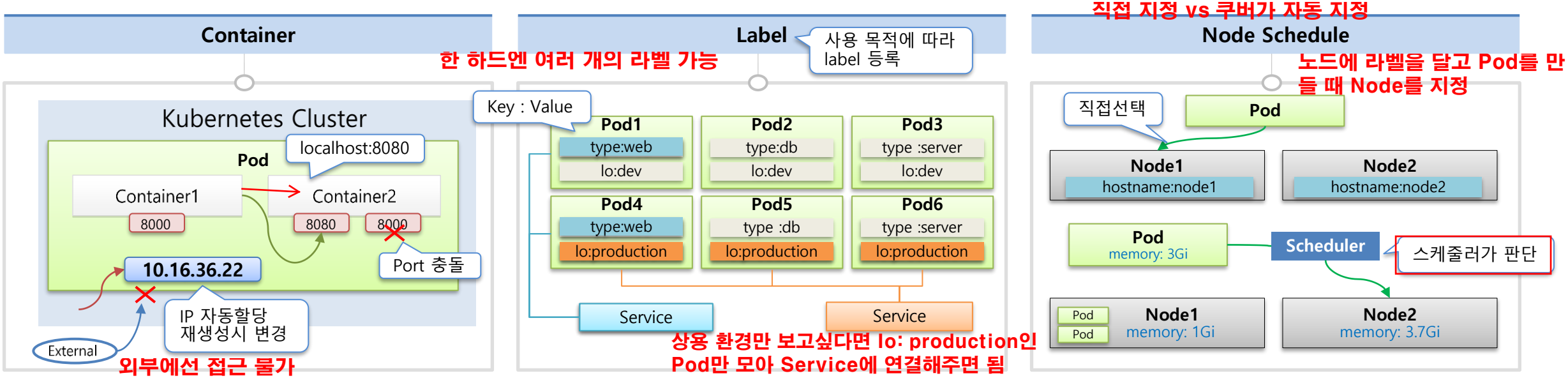
5. Object - Pod




```yaml
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
```

서비스를 만들 때 라벨을 이용해서 묶음

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

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

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

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

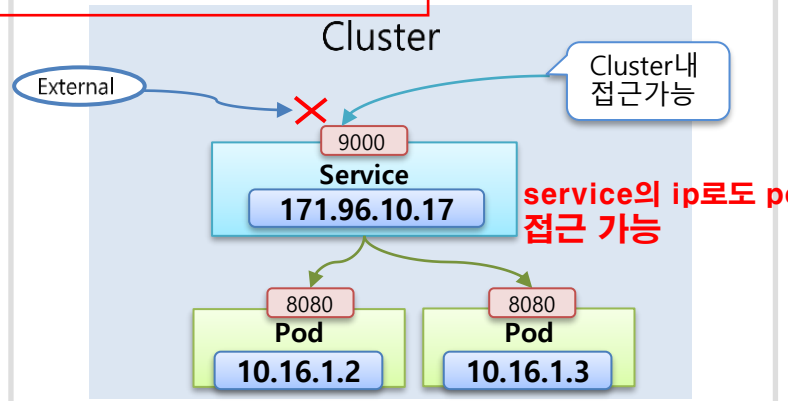
Memory : 초과시 Pod 종료 시킴
Cpu : 초과시 request로 낮춤
Over시 종료되지 않음

5. Object - Service

- 서비스 디버깅
- 내부 트래픽/대쉬보드
- 인증된 사용자 연결

ClusterIP

클러스터 내에서만 접근 가능



service의 ip로도 pod에 접근 가능

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

9090 -> 8080

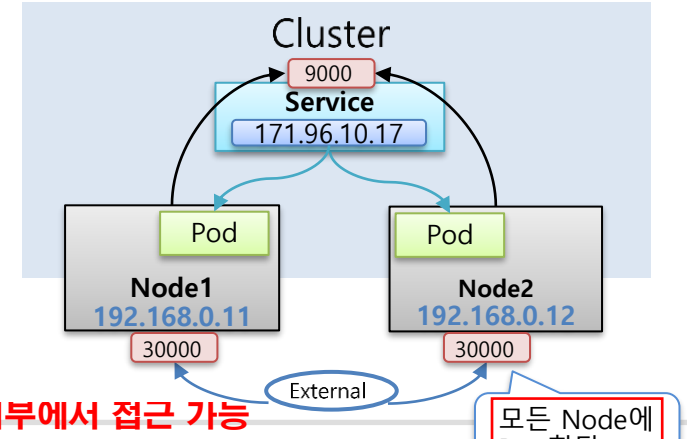
기본값: clusterip임

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

service를 통해 접근하는 이유
- pod는 죽으면 언제든 재 생성 된다. 그때 ip가 바뀌기 때문

- 데모나 임시 연결용
- 프로덕션 환경(X)

NodePort



외부에서 접근 가능

모든 Node에 Port할당

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

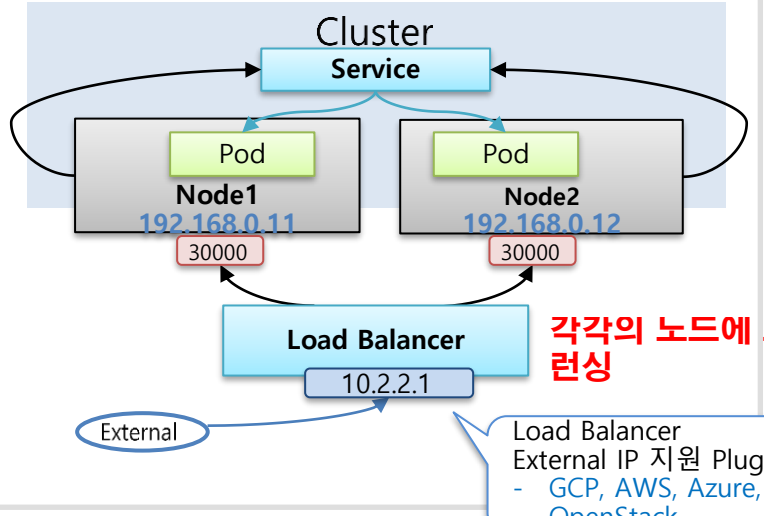
30000~32767

externalTrafficPolicy: Local

이게 없으면 Node1에 접근해도 Node1, 2 다 갈 수 있음 => 로드밸런싱
있으면 접근한 Node만 접근 가능

- 외부 시스템 노출용

Load Balancer



각각의 노드에 로드밸런싱

Load Balancer
External IP 지원 Plugin
- GCP, AWS, Azure, OpenStack

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

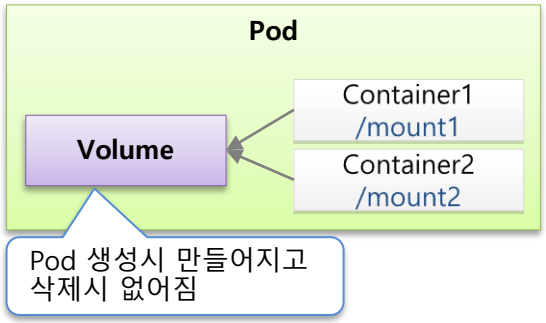
5. Object - Volume

최초 Admin이 PV를 만들고 사용자가 PVC를 만들면
쿠버네티스가 PVC내용에 맞는 적절한 PV에 연결해줌
그 다음 Pod를 만들 때 PVC를 사용

일시적인 사용 목적에 의한 데이터만 넣기

emptyDir

컨테이너 1,2 가 서로 파일을 주고 받을 필요 없이 mount된 걸 쓰면 됨



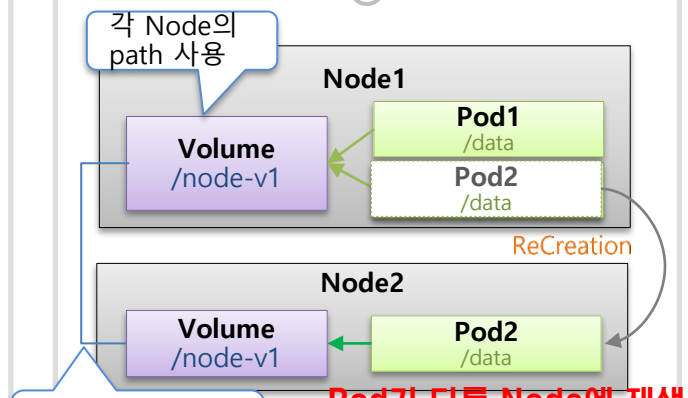

```
apiVersion: v1
kind: Pod
metadata:
 name: pod-volume-1
spec:
 containers:
 - name: container1
   image: tmkube/init
   volumeMounts:
   - name: empty-dir
     mountPath: /mount1
 - name: container2
   image: tmkube/init
   volumeMounts:
   - name: empty-dir
     mountPath: /mount2
 volumes:
 - name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

이게 같아야함

mountPath는 다르지만 name이 같아서 공유

hostPath

Pod가 죽어도 괜찮



Pod가 다른 Node에 재생성 시켜준다면 볼륨 연결 실패

```
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
```

DirectoryOrCreate로 하면 만들어짐

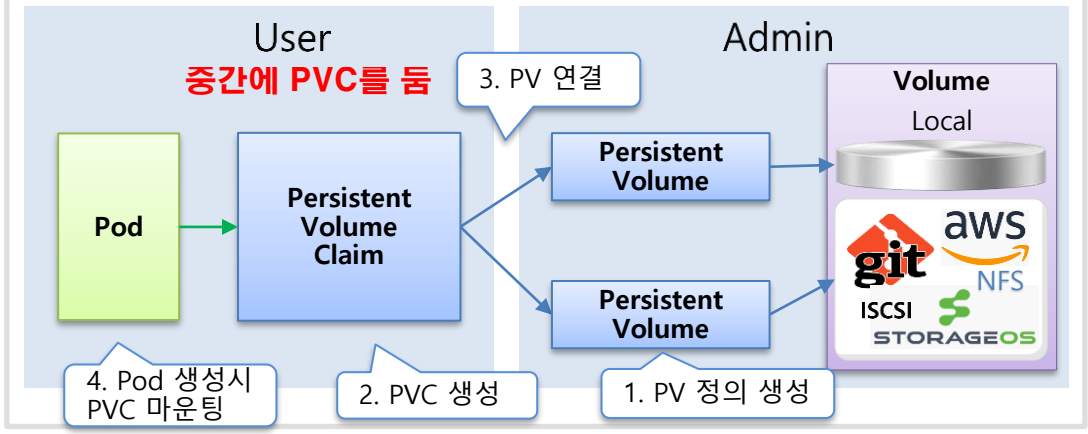
사전에 해당 Node에 경로가 있어야함

Pod가 만들어지기 전에 생성되있어야 함

PVC / PV

서비스 담당자가 관리

쿠버 운영자가 관리




```
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
```

Pod를 만들때 만들어놓은 pvc연결

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

읽기 쓰기

용량 1기가

현재 만들어진 PV들 중 선택이 됨

```
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]}
```

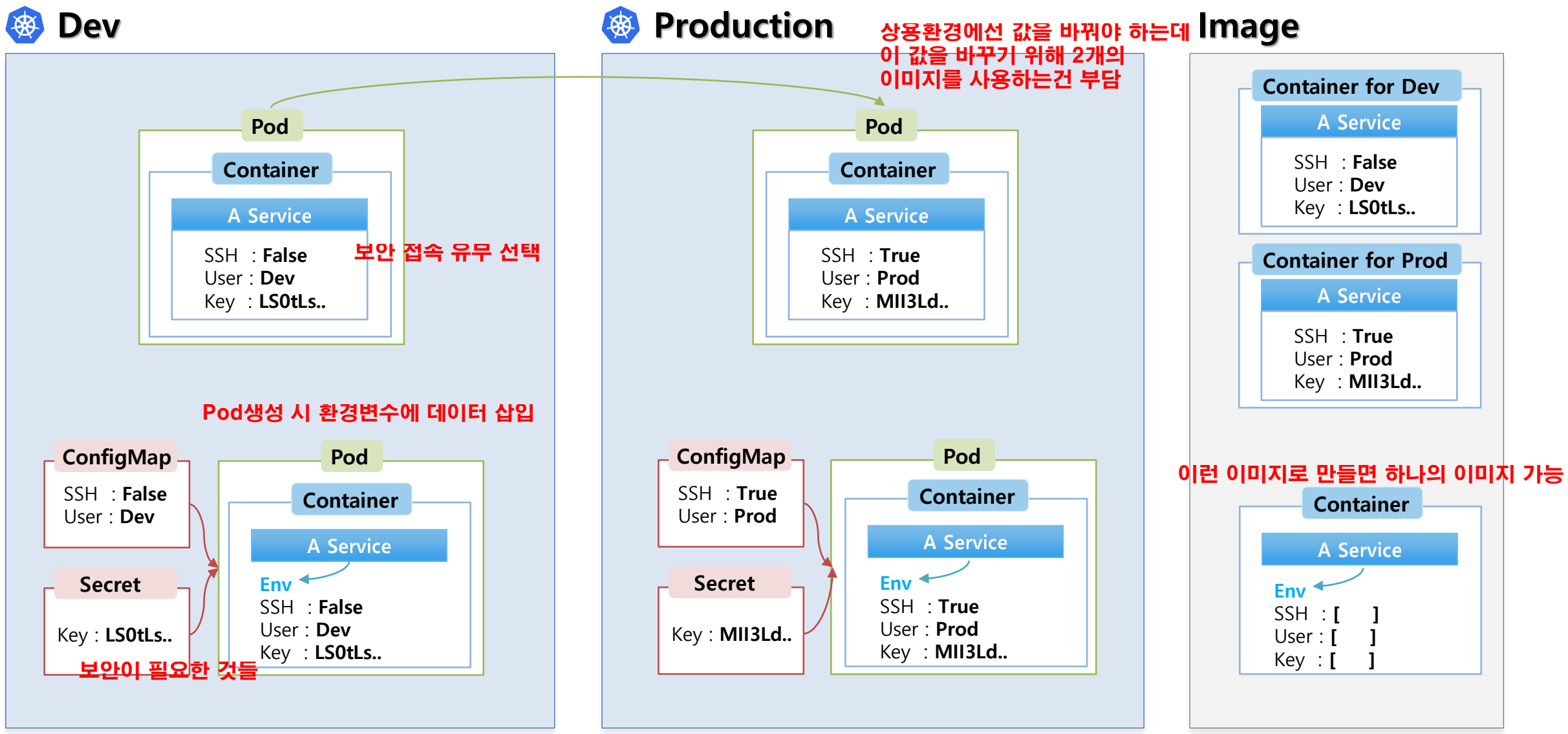
이 PV에 연결되는 Pod들은 node1에 만들어짐

Pod가 해당 Node에 만들어짐

```
nfs:
  serve
  path:
iscsi:
  targe
  iqn: i
  lun: 0
  fsTyp
  read
  chap
gitRep
  repo
  revis
  direc
```

5. Object - ConfigMap, Secret

환경에 따라 변하는 값들을 외부에서 설정



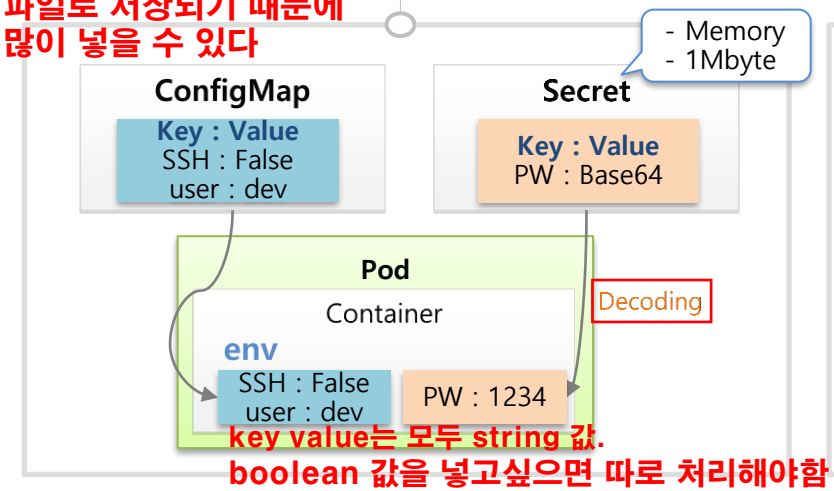
5. Object - ConfigMap, Secret

추후에 ConfigMap에 데이터가 변경된다면
Env는 Pod에 반영되지 않는 반면 Mount는 반영된다.

Env (Literal)

파일로 저장되기 때문에
많이 넣을 수 있다

Secret은 메모리에 저장됨



key value는 모두 string 값.
boolean 값을 넣고싶으면 따로 처리해야함

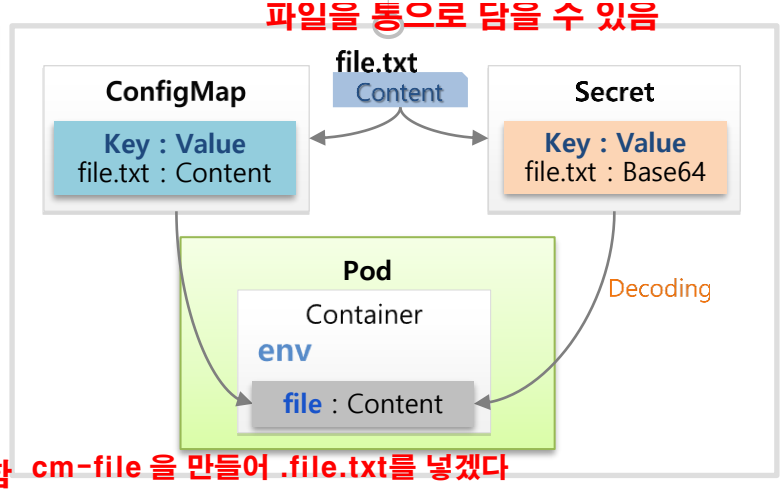
```
apiVersion: v1
kind: ConfigMap
metadata:
 name: cm-dev
data:
 SSH: False
 User: dev
```

```
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)

파일을 통으로 담을 수 있음



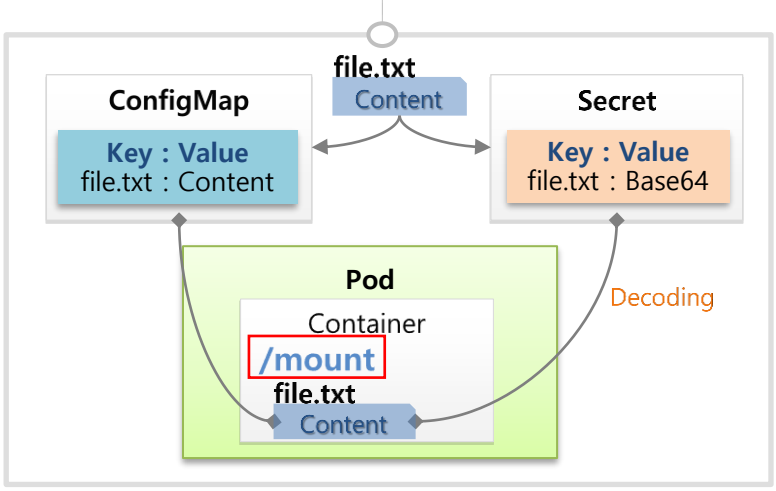
cm-file 을 만들어 .file.txt를 넣겠다

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

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

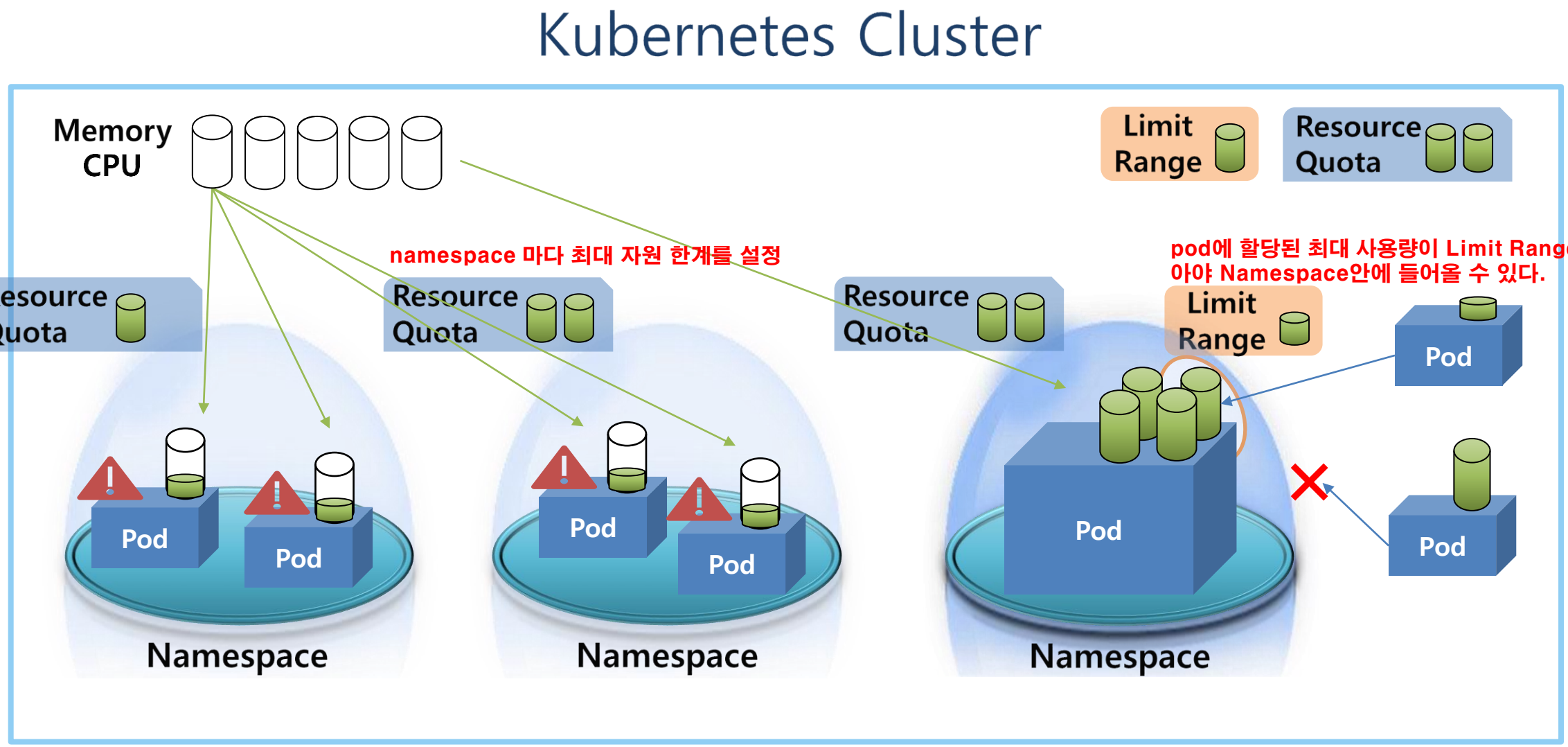
```
apiVersion: v1
kind: Pod
metadata:
  name: file
spec:
  containers:
    - name: container
      image: tmkube/init
      env:
        - name: file
          valueFrom:
            configMapKeyRef:
              name: cm-file
              key: file.txt
```

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
```

5. Object - Namespace, ResourceQuota, LimitRange



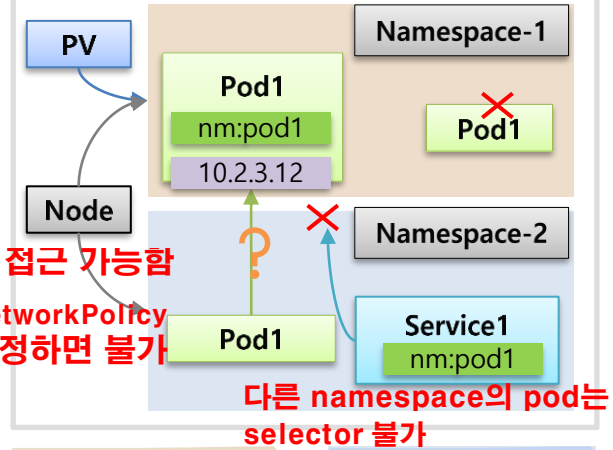
5. Object - Namespace, ResourceQuota, LimitRange

다른 namespace와의 자원 분리 가능

하나의 namespace엔 하나의 limitrange를 지정하자

Namespace

하나의 namespace에서 중복pod이름 불가




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

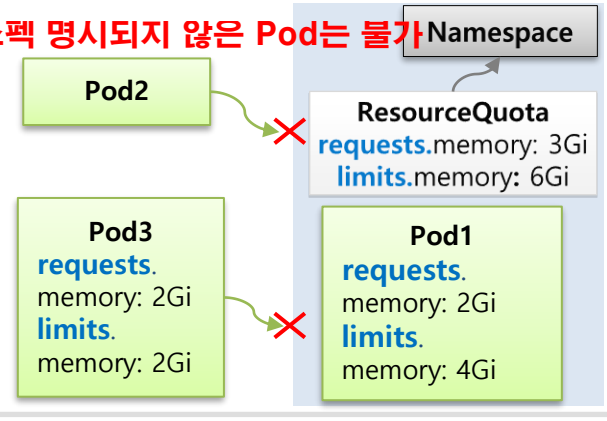
```
apiVersion: v1
kind: Namespace
metadata:
  name: nm-2
```

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

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

ResourceQuota

스펙 명시되지 않은 Pod는 불가



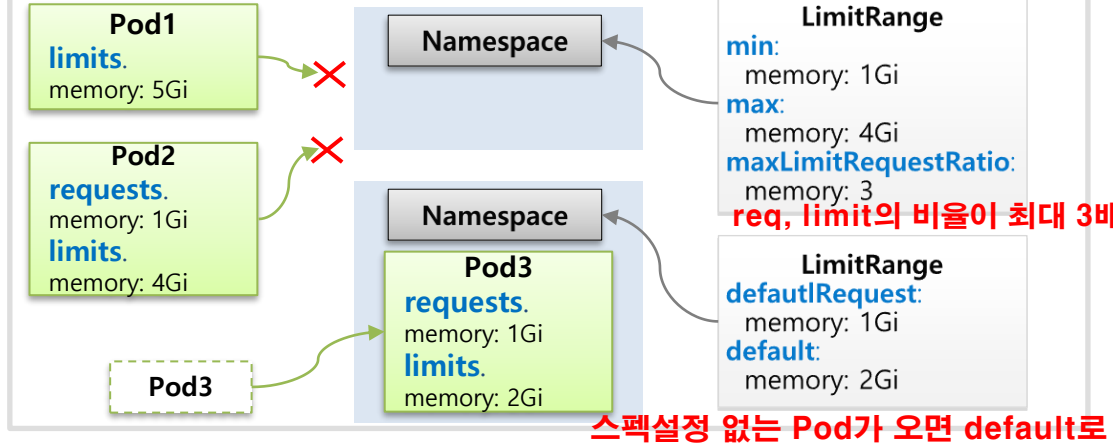

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: rq-1
  namespace: nm-1
spec:
  hard:
    requests.memory: 3Gi
    limits.memory: 6Gi
```

```
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

각각의 Pod마다 검사



req, limit의 비율이 최대 3배

스펙설정 없는 Pod가 오면 default로 할당

```
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
```

service생성 시 nodeport, host-path 볼륨 등은 namespace에 분리되지 않는다.