1. 리소스 관리

- ❖Pod 리소스 관리 개요
- ❖limits와 requests
- ❖QoS (Quality of Service)
- ❖CPU와 메모리에 대한 리소스 관리
- ❖ResourceQuota

1.1 Pod 리소스 관리 개요

❖Pod, Container의 리소스 관리를 하지 않으면?

- Container는 Worker Node의 CPU, Memory를 최대로 사용할 수 있음
  --〉 Worker Node의 Resource가 고갈될 수 있음
- 따라서 Pod의 리소스 사용량을 명시적으로 설정해야 함

❖설정하는 방법

- CPU, Memory 에 대해 Limit, Request를 설정함
- ResourceQuota 설정
- LimitRange 설정

1.2 k8s metric-server 설치

❖k8s metric server란?

- k8s의 node, pod가 리소스를 얼마나 사용하고 있는지를 확인할 수 있는 모니터링 도구
- metric server 를 설치했다면 kubectl top node, kubectl top pod 명령어를 실행할 수 있음

❖설치 방법

- 설치 명령어

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

- 설치 후 확인해보면 실행이 정상적이지 않음
- 문제 원인 해결 : metric-server가 모든 노드의 유효한 tls 인증서를 확인할 수 없어서 발생한 문제
- 문제 해결 : insecure-tls 를 허용함

```
kubectl edit deployment -n kube-system metrics-server
```

```
spec:
  containers:
  - args:
    - --cert-dir=/tmp
    - --secure-port=10250
    - --kubelet-insecure-tls
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --kubelet-use-node-status-port
    - --metric-resolution=15s
```

1.2 limits와 requests

❖개념

- limits : 컨테이너에게 허용할 최대 리소스(CPU, Memory) 용량 제한 값
- requests : 컨테이너에게 보장해야 할 리소스 용량

❖ requests 예제 : resource1.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: limit1
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodeapp
  template:
    metadata:
      name: nodeapp
      labels:
        app: nodeapp
```

```
    spec:
      containers:
      - name: nodeapp-slim
        image: stepanowon/nodeapp-slim:1.0.0
        ports:
        - containerPort: 8080
        resources:
          limits:
            memory: "100Mi"
            cpu: "32m"

# 메모리
#  - Mi : 1000 Mi byte --> 1 Gi Byte
#  - M :   1024 MB --> 1 GB
# CPU
#  - m : milli core,  1000 milli core --> 1 (CPU)
```

1.2 limits와 requests

❖requests 예제 : resource1.yaml (이어서)

- LoadBalancer 생성 : kubectl apply -f nodeapp-lb.yaml
  - External IP 확인 : 예시) 192.168.56.51
- Deployment 생성 : kubectl apply -f resource1.yaml
- 생성된 리소스 확인

```
stepano@stepanowon:~/k8s/resource$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/resource1-67866ccbd4-4c9qd   1/1     Running   0          82s
pod/resource1-67866ccbd4-cf76t   1/1     Running   0          84s

NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
service/kubernetes   ClusterIP      10.96.0.1      <none>          443/TCP        5d17h
service/nodeapp-lb   LoadBalancer   10.99.20.172   192.168.56.51   80:32621/TCP   163m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/resource1    2/2     2            2           12m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/resource1-67866ccbd4    2         2         2       84s
replicaset.apps/resource1-7c7ffb85bf    0         0         0       12m
```

1.2 limits와 requests

❖생성된 Pod의 request, limit 확인

- kubectl describe pods 파드명
- custom 열로 조회

```
kubectl get pods -o custom-columns="Name:metadata.name,\
Requet-CPU:spec.containers[*].resources.requests.cpu,\
Request-MEM:spec.containers[*].resources.requests.memory,\
Limit-CPU:spec.containers[*].resources.limits.cpu,\
Limit-MEM:spec.containers[*].resources.limits.memory"
```

```
stepano@stepanowon:~/k8s/resource$ kubectl get pods -o custom-columns="Name:metadata.name,\
Requet-CPU:spec.containers[*].resources.requests.cpu,\
Request-MEM:spec.containers[*].resources.requests.memory,\
Limit-CPU:spec.containers[*].resources.limits.cpu,\
Limit-MEM:spec.containers[*].resources.limits.memory"
Name                          Requet-CPU   Request-MEM   Limit-CPU   Limit-MEM
resource1-76547d84b6-sjsmz    32m          100Mi         32m         100Mi
resource1-76547d84b6-tswhr    32m          100Mi         32m         100Mi
```

- limits 만을 설정하면 requests 는 limits와 동일한 값으로 자동 설정됨

1.2 limits와 requests

❖실제 리소스 사용량 확인 : 테스트 완료후 deployment 삭제할 것

```
stepano@stepanowon:~/k8s/resource$ kubectl top node
NAME       CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master     386m         19%    1871Mi          49%
worker1    36m          3%     1125Mi          60%
worker2    38m          3%     1025Mi          54%
stepano@stepanowon:~/k8s/resource$ kubectl top pod
NAME                              CPU(cores)   MEMORY(bytes)
resource1-67866ccbd4-4c9qd        1m           47Mi
resource1-67866ccbd4-cf76t        1m           47Mi
```

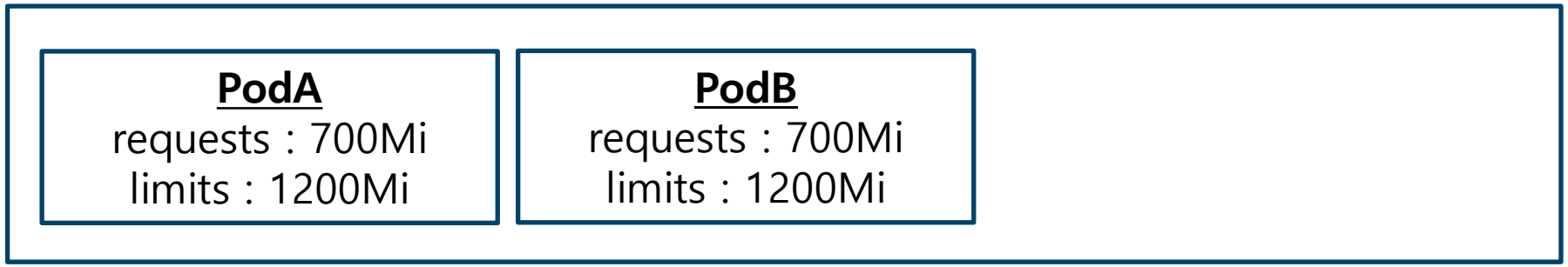
- Pod의 리소스 사용량이 limits 에 설정된 값보다 작은데?
  - limits은 리소스 최대 사용량을 제한하기 위한 값이며, 실제 리소스 사용량을 의미하는 것은 아님
- 만일 requests와 limit을 다르게 설정한다면?
  - requests는 노드의 리소스에 영향을 받지만 limits은 영향을 받지 않음
    - 노드의 리소스보다 더 많은 limits을 설정할 수 있음
    - limits 는 최대 사용량의 제한 값이지 실제 사용량을 아니기 때문에...
    - Overcommit 상태 : 노드에 스케줄링된 Pod의 limits 메모리의 합계 〉 노드의 사용 가능한 메모리 공간
  - 반면 requests는 노드의 리소스에 영향을 받음
    - 보장하는 용량이므로...

1.2 limits와 requests

❖리소스 테스트

- Node의 메모리 : 2000 Mi
- Node 상태 예시 1

노드의 사용가능한 메모리: 2000Mi



- requests의 합계 : 1200Mi 〈 2000Mi
- limits의 합계 : 2100Mi 〉 2000Mi ---〉 이러한 상황이 가능
- limits의 값이 실제 리소스 사용량이라고 할 수는 없음.
- 하지만 언제든지 requests 설정 값을 넘어서 limits 보다 작은 값으로는 burstable(동적으로 확장 가능)함.
  - 위 그림에서 limits의 합계는 노드의 사용가능한 메모리를 초과하지만 Pod는 정상적으로 생성됨.
  - requests의 합계는 사용가능한 메모리보다 반드시 작아야 함. (이유: 보장되어야 하는 용량이므로)

1.2 limits와 requests

❖리소스 테스트(이어서)

- worker node의 사용가능한 리소스 확인
  - kubectl describe node 노드명 : ex) kubectl describe node worker1
    - Capacity : 노드의 전체 리소스 용량
    - Allocatable : 노드에서 사용가능한 메모리 용량
  - 테스트 시 용량 확인 : 1913036Ki -〉 1913Mi
- resource2.yaml : 메모리만 확인, 4 replica --〉 worker가 3개라면 replicas를 6으로 설정

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: resource2
  namespace: default
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nodeapp
  template:
    metadata:
      name: nodeapp
      labels:
        app: nodeapp
```

```
    spec:
      containers:
      - name: nodeapp-slim
        image: stepanowon/nodeapp-slim:1.0.0
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "700Mi"
          limits:
            memory: "1200Mi"
```

1.2 limits와 requests

❖리소스 테스트(이어서)

- 리소스 생성 후 리소스 사용량 확인

```
stepano@stepanowon:~/k8s/resource$ kubectl apply -f resource2.yaml
deployment.apps/resource2 created
stepano@stepanowon:~/k8s/resource$ kubectl get pods -o wide
NAME                         READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
resource2-5568cf8889-g928d   1/1     Running   0          38s   10.244.235.174   worker1   <none>           <none>
resource2-5568cf8889-x7jw8   1/1     Running   0          38s   10.244.189.113   worker2   <none>           <none>
resource2-5568cf8889-xfkxx   1/1     Running   0          38s   10.244.189.112   worker2   <none>           <none>
resource2-5568cf8889-z66wp   1/1     Running   0          38s   10.244.235.173   worker1   <none>           <none>
stepano@stepanowon:~/k8s/resource$ kubectl top node
NAME      CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master    559m         27%    1941Mi          50%
worker1   167m         16%    1121Mi          60%
worker2   206m         20%    1089Mi          58%
```

- 검토
  - limits 메모리의 합계가 node 의 사용가능한 메모리 합계보다 크지만 Pod가 정상적으로 생성되었음
  - requests의 합계가 node의 사용가능한 메모리를 초과하는 경우 생성불가함
  - 따라서 node의 replica를 늘리면 더이상 Pod가 생성되지 못하고 Pending 상태가 될 것임

1.2 limits와 requests

❖리소스 테스트(이어서)

- resource2.yaml에서 replica 갯수를 1증가시켜 5로 설정 후 적용
  - kubectl apply -f resource2.yaml
  - worker node를 3개로 사용하고 있다면 6에서 7 또는 8로 변경
- 추가된 Pod가 Pending 상태인 것을

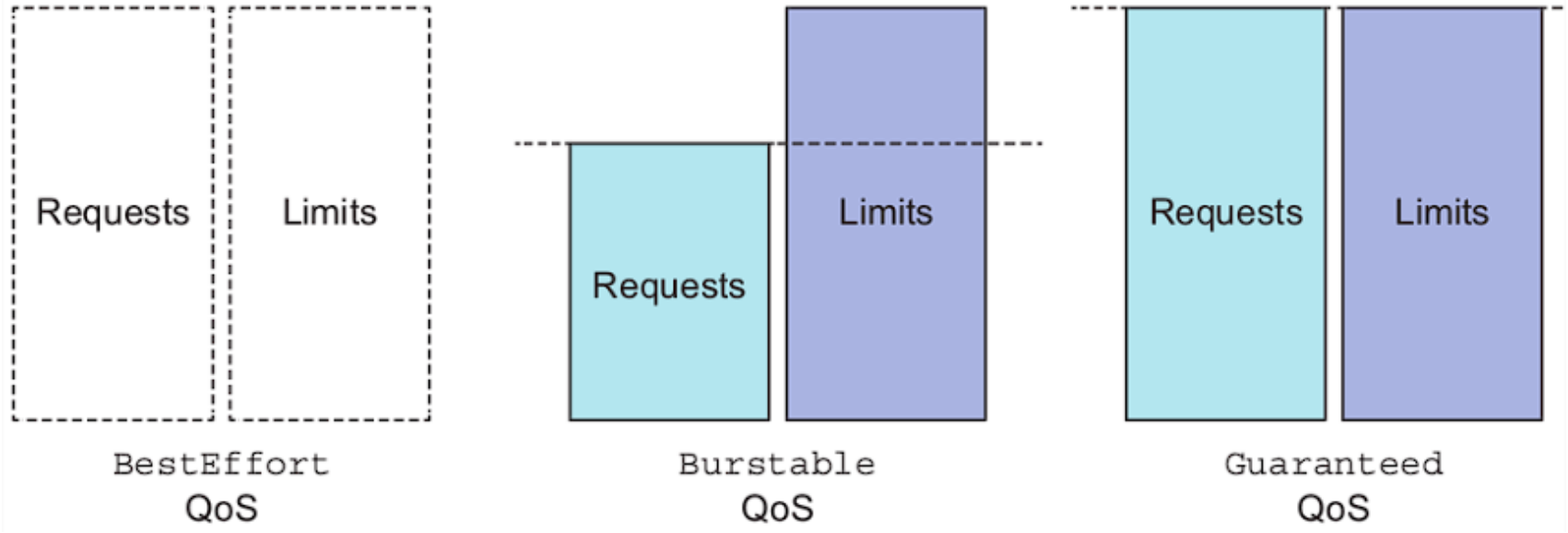
```
stepano@stepanowon:~/k8s/resource$ kubectl get pods -o wide
NAME                         READY   STATUS    RESTARTS   AGE     IP               NODE      NOMINATED NODE   READINESS GATES
resource2-5568cf8889-9r6gb   1/1     Running   0          6m46s   10.244.189.117   worker2   <none>           <none>
resource2-5568cf8889-dp5n9   0/1     Pending   0          4m54s   <none>           <none>    <none>           <none>
resource2-5568cf8889-g7qlf   1/1     Running   0          6m46s   10.244.189.118   worker2   <none>           <none>
resource2-5568cf8889-rfl4j   1/1     Running   0          6m46s   10.244.235.178   worker1   <none>           <none>
resource2-5568cf8889-zzjgl   1/1     Running   0          6m46s   10.244.235.179   worker1   <none>           <none>
```

- pending 상태인 이유
  - requests memory 의 합계가 노드의 사용가능한 메모리보다 크기 때문

1.3 QoS

❖k8s의 QoS(Quality of Service)란?

- Pod의 서비스 품질을 의미하며, 중요도가 높은 서비스의 품질을 보장하기 위해 리소스의 용량을 설정하는 것을 말함
- QoS의 종류, 우선순위 : Best Effort 〈 Burstable 〈 Guaranteed



설정(X) **requests 〈 limits** **requests = limits**

출처 : https://libert.xyz/posts/pods-qos/

1.3 QoS

❖ QoS 확인 방법

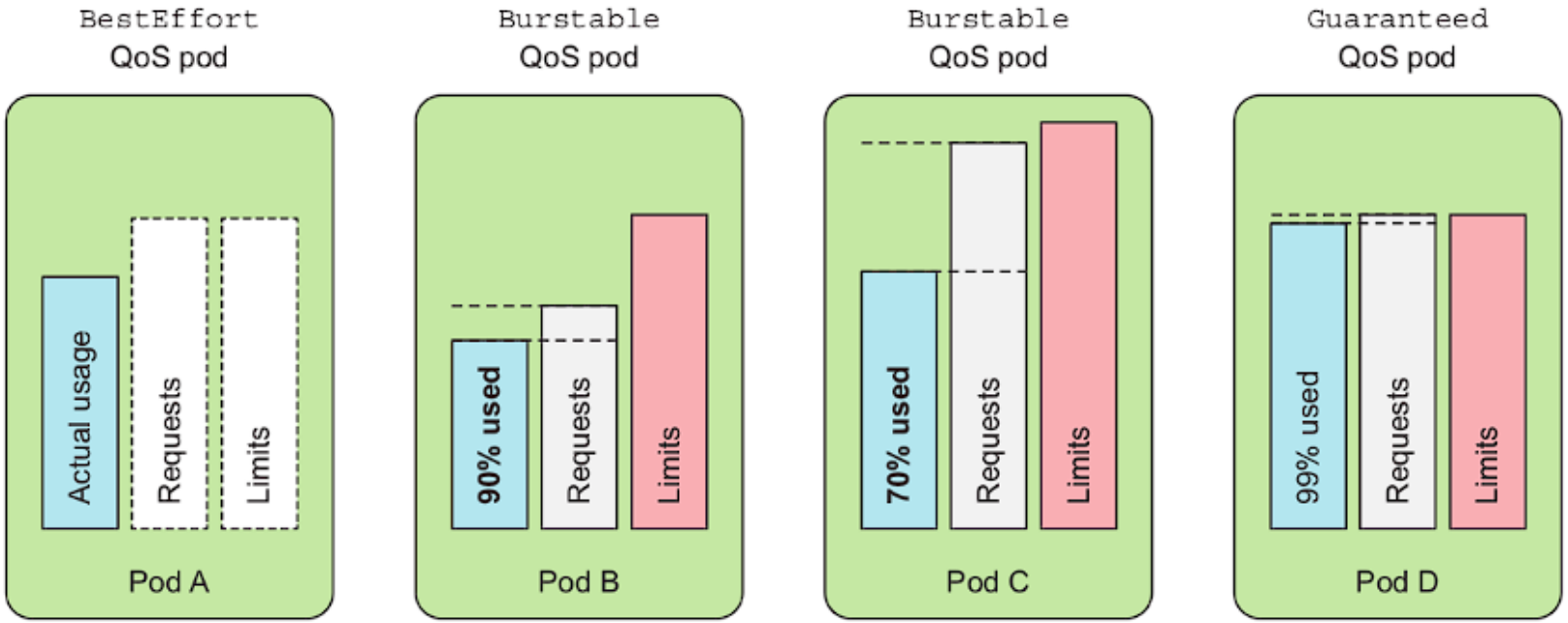
- kubectl describe pods 파드명 -n 네임스페이스

```
Name:                 limit1-76547d84b6-bdkjv
Namespace:            default
......(생략)
Volumes:
  kube-api-access-s6wrw:
    Type:                     Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:   3607
    ConfigMapName:            kube-root-ca.crt
    ConfigMapOptional:        <nil>
    DownwardAPI:              true
QoS Class:                    Guaranteed
Node-Selectors:               <none>
Tolerations:                  node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                              node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  ......(생략)
```

1.3 QoS

❖QoS가 끼치는 영향

- k8s 내부에서 리소스가 부족한 경우 QoS에 따라 어떤 Pod를 종료시킬지를 결정함
- 가장 우선순위가 낮는 Pod부터 종료됨.
  - Best Effort – Burstable 순으로 Pod가 종료되며, Guaranteed는 시스템이 더이상 종료할 Pod가 없는 경우에 종료됨



출처 : https://libert.xyz/posts/pods-qos/

1.4 CPU와 메모리에 대한 리소스 관리

❖ 압축 가능 리소스와 압축 불가능 리소스

- Compressible Resource
  - 대표적인 리소스가 CPU, Network
  - 만일 경합이 발생하면(Overcommit 상태가 되면) Throttling이 발생하긴 하지만 Time Slicing을 통해 분배가 가능함
- Incompressible Resource
  - 대표적인 리소스가 Memory, Storage
  - 리소스가 부족해지면 프로세스가 더이상 기다릴 수 없음을 의미함
  - OOM(Out of Memory) Killer : 메모리가 부족할 경우 특정 Pod를 강제로 종료시킴
- Limits 설정값을 초과한 경우
  - CPU : Pod가 종료되지는 않지만 CPU Throttling이 발생하므로 지연이 발생하거나 느려짐
  - Memory : Limits 값을 넘어설 수 없으므로 Pod는 OOM Killer에 의해 강제로 종료됨

1.5 Resource Quota와 Limit Range

❖Resource Quota란?

- namespace 단위로 총 리소스 사용을 제한하는 제약 조건을 제공
- 리소스 유형별로 namespace에서 만들 수 있는 오브젝트의 갯수와 namespace의 리소스가 사용할 수 있는 총 컴퓨팅 리소스의 양을 제한함.
- 생성되는 pod가 resource-quota의 적용을 받으려면 pod spec에서 반드시 resource 설정을 해야하고 cpu, memory 모두 포함해야 함

```
### resource-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: resource-quota-sample
  namespace: default
spec:
  hard:
    pods: 4
    requests.cpu: 2
    requests.memory: 2Gi
    limits.cpu: 4
    limits.memory: 4Gi
```

1.5 Resource Quota와 Limit Range

❖테스트용 deployment 배포

```
### nodeapp-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp-deploy
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodeapp
  template:
    metadata:
      name: nodeapp
      labels:
        app: nodeapp
    spec:
      containers:
      - name: nodeapp
        image: stepanowon/nodeapp-slim:1.0.0
        ports:
        - containerPort: 8080
```

```
        resources:
          requests:
            memory: "100Mi"
            cpu: "64m"
          limits:
            memory: "200Mi"
            cpu: "128m"
```

```
## 다음 명령 실행
kubectl apply -f resource-quota.yaml
kubectl apply -f nodeapp-deploy.yaml

## Pod 정상 배포 확인 후 다음 명령 실행
kubectl scale deployment nodeapp-deploy --replicas=5

## Pod가 5개 아닌 4개인 것을 확인
## ResourceQuota에 의해서 제한된 것

## 리소스 정리
kubectl delete -f resource-quota.yaml
kubectl delete -f nodeapp-deploy.yaml
```

1.5 Resource Quota와 Limit Range

❖Limit Range란?

- 지정한 Namespace의 리소스에 대한 requests, limits 값을 설정할 수 있음
- 설정 항목
  - default : 기본 limits
  - defaultRequests : 기본 requests
  - max : 최대 리소스 값
  - min : 최소 리소스 값
  - maxLimitRequestRatio : limits / requests 의 비율
- 설정 대상(type)
  - Container : default, defaultRequests, max, min, maxLimitRequestRatio
  - Pod : max, min, maxLimitRequestsRatio 를 설정할 수 있음
- 신규로 파드를 생성할 때 사용되므로 기존에 실행되고 있는 Pod에는 영향을 주지 않음

```
### limitrange.yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: limitrange1
  namespace: default
spec:
  limits:
  - type: Container
    default:                  # 기본limits
      memory: 128Mi
      cpu: 200m
    defaultRequest:           # 기본 requests
      memory: 64Mi
      cpu: 100m
    maxLimitRequestRatio:     # limits/requests 비율값
      memory: 2
      cpu: 2
    max:                      # 최대 리소스
      memory: 256Mi
      cpu: 4000m
    min:                      # 최소 리소스
      memory: 32Mi
      cpu: 50m
```

1.5 Resource Quota와 Limit Range

❖Limit Range 테스트

```
## nodeapp-deploy2 배포 : 이것은 nodeapp-deploy.yaml에서 resource 설정을 제거한 것
kubectl apply -f nodeapp-deploy2.yaml

## Pod 리스트 조회 후 생성된 pod 이름으로 describe 명령 수행하여 QoS가 best effort임을 확인
kubectl get pods
kubectl describe pods nodeapp-deploy2-xxxxxxx-yyyy

## LimitRange 리소스 생성하고 deploymen의 replcas 값 변경
kubectl apply -f limitrange.yaml
kubectl scale deployment nodeapp-deploy2 --replicas=3

## 가장 최근에 생성된 pod의 QoS 확인
kubectl get pods 명령 실행 후 AGE 필드 확인
kubectl describe pods nodeapp-deploy2-zzzzzz-kkkk

## 리소스 정리
kubectl delete -f nodeapp-deploy2.yaml
kubectl delete -f limitrange.yaml
```