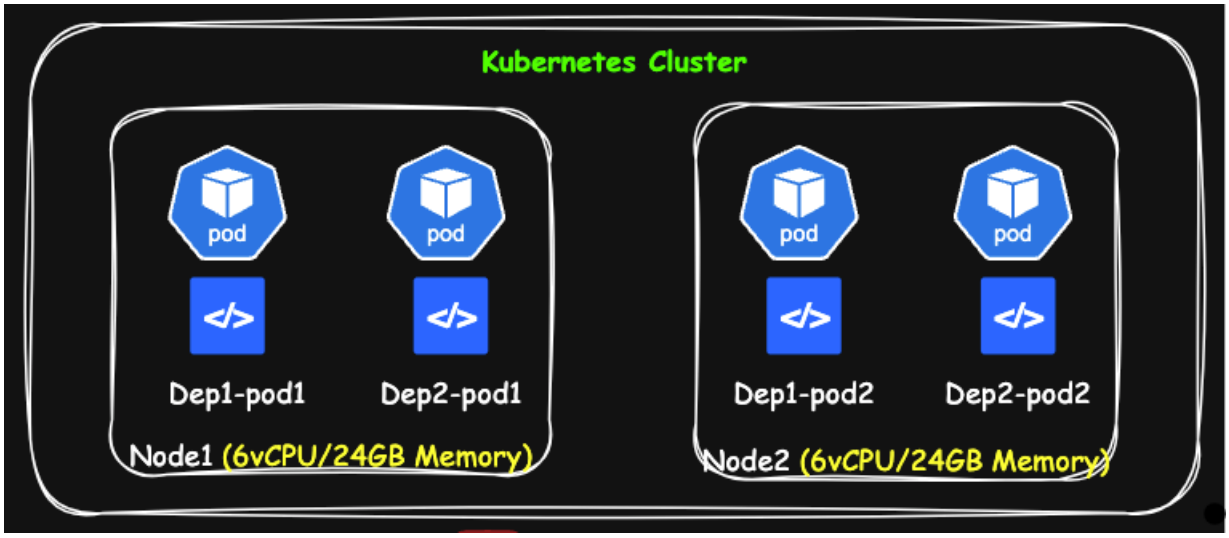
REQUESTS, LIMITS & LIMIT-RANGES



Everything runs smoothly until deployment-2 malfunctions. A code bug causes its pods to consume excessive CPU and memory, starving deployment-1 resources and creating a noisy-neighbor problem.

This is where the concept of requests and limits acts as a safeguard.

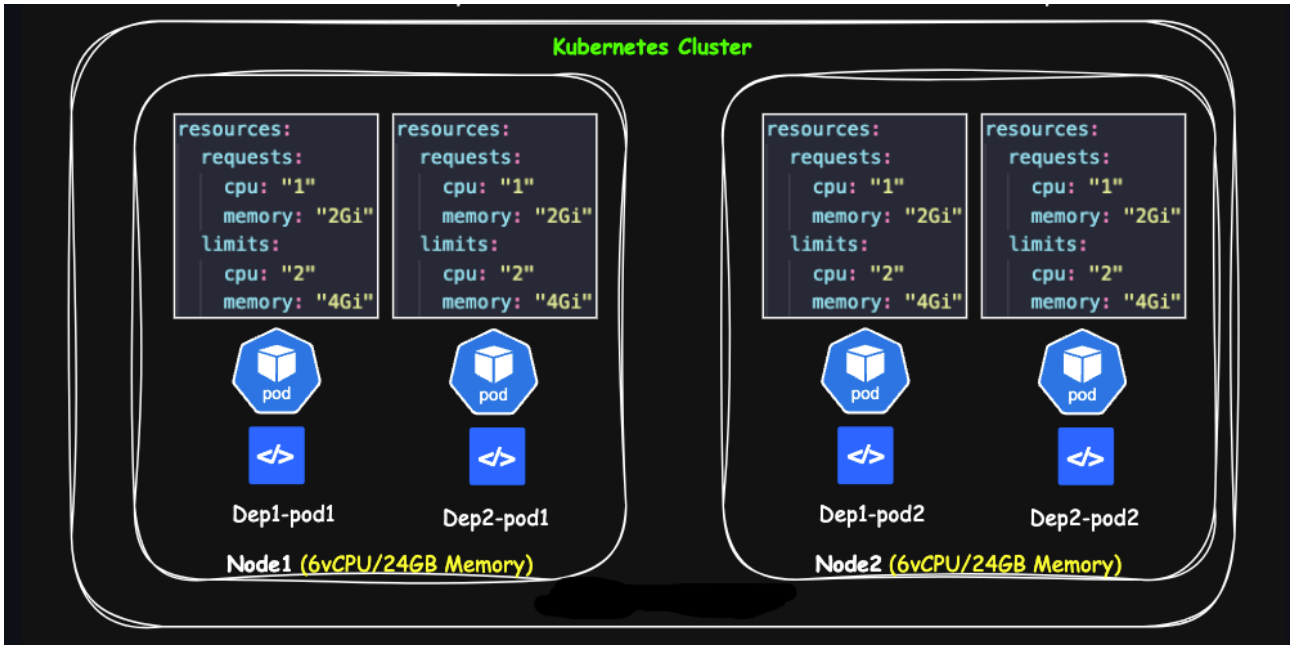
Request: Minimum guaranteed resources (typically CPU & Memory) for a container.

Limits: Maximum resources(typically CPU & Memory) a container can use as long as the resources are available from the node.

Benefits of Using Requests and Limits:

1. Efficient Resource Allocation
2. Avoidance of resource Starvation - Prevents one workload from hogging all the resources.
3. Cluster Stability
4. Mitigation of Noisy Neighbour problem - A container that suddenly consumes excessive node resources becomes a noisy neighbor, and limits help prevent it from impacting other workloads.

Note: It is recommended to define Requests and Limits for each container within a pod.



→ If a node has 6 CPUs and 24 GB of memory, and six pods deployed, each request 1 CPU and 4 GB, the node becomes fully allocated. A seventh pod cannot be scheduled because no remaining resources are available.

→ Pods can use the amount of resources they request, and if they need more, they can consume up to their defined limit—provided the node has available capacity.

What happens when a container goes beyond resource limits?

CPU: The **KERNEL** THROTTLES the CPU.

Memory: The **KERNEL** may Terminate the pod.

- **Memory Pressure (no memory available):** The pod is **OOM**-killed.
- **No Memory Pressure:** The pod continues using memory beyond its limit.

→ Lets Explore the concept of Request and Limits Practically!

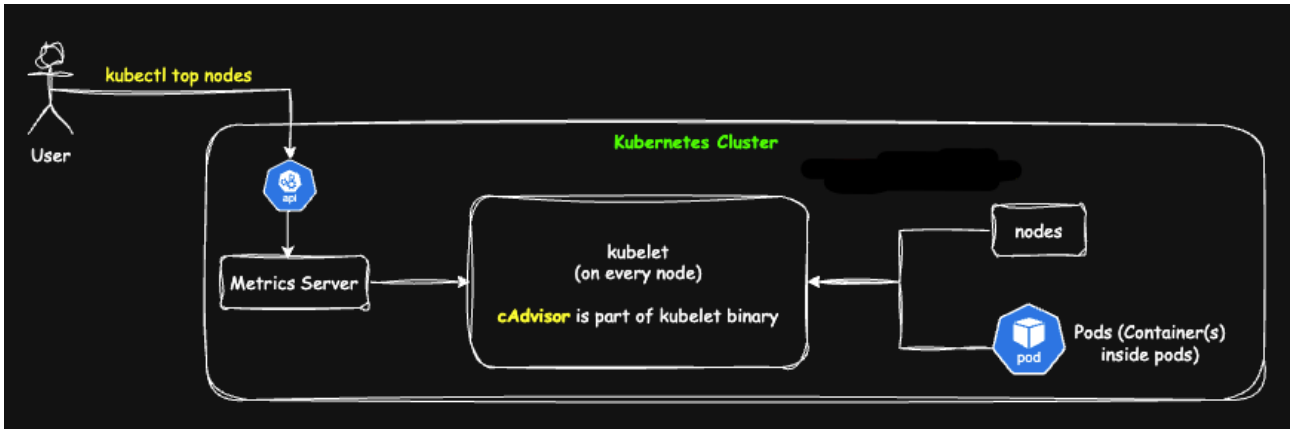
First, we need to set up the metrics server on our cluster. We can do this by downloading the required manifest from the Kubernetes GitHub repository:

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

And we have to add this flag to the deployment manifest in this file:

```
--kubelet-insecure-tls
```

```
spec:
  containers:
    - args:
        - --cert-dir=/tmp
        - --secure-port=10250
        - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
        - --kubelet-use-node-status-port
        - --metric-resolution=15s
        - --kubelet-insecure-tls
      image: registry.k8s.io/metrics-server/metrics-server:v0.8.0
```



→ The Kubelet binary includes a component called **cAdvisor** (Container Advisor), which exposes an API that collects metrics from nodes and pods and provides them to the Metrics Server.

Lets deploy metric server on the cluster:

```
kubectl apply -f components.yaml
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl apply -f components.yaml
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

```
kubectl get pods -n kube-system
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl get pods -n kube-system
NAME                                                READY   STATUS    RESTARTS   AGE
coredns-7c65d6cfc9-89rxf                            1/1     Running   0          55m
coredns-7c65d6cfc9-w4px7                            1/1     Running   0          55m
etcd-rayeez-cluster-control-plane                   1/1     Running   0          56m
kindnet-c6zxx                                       1/1     Running   0          55m
kindnet-kx97f                                       1/1     Running   0          55m
kindnet-snjtk                                       1/1     Running   0          55m
kube-apiserver-rayeez-cluster-control-plane         1/1     Running   0          56m
kube-controller-manager-rayeez-cluster-control-plane   1/1     Running   0          56m
kube-proxy-ns5l4                                    1/1     Running   0          55m
kube-proxy-sspcd                                    1/1     Running   0          55m
kube-proxy-zljdz                                    1/1     Running   0          55m
kube-scheduler-rayeez-cluster-control-plane         1/1     Running   0          56m
metrics-server-bf688598-gp85q                       1/1     Running   0          3m16s
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$
```

Lets verify resource usage by nodes and pods:

```
kubectl top nodes
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl top nodes
NAME                           CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
rayeez-cluster-control-plane   191m         1%       818Mi           22%
rayeez-cluster-worker          32m          0%       212Mi           5%
rayeez-cluster-worker2         26m          0%       177Mi           4%
```

```
kubectl top pods
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl top pods
NAME                                  CPU(cores)   MEMORY(bytes)
frontend-deploy-6df94c895b-fwsps      12m          10Mi
frontend-deploy-6df94c895b-gpspl      0m           10Mi
frontend-deploy-6df94c895b-msmvx      14m          10Mi
frontend-deploy-6df94c895b-rlr4v      0m           10Mi
frontend-deploy-6df94c895b-tmkjp      12m          10Mi
```

```
kubectl describe nodes rayeez-cluster-worker2
```

```
Namespace                   Name                                CPU Requests  CPU Limits  Memory Requests  Memory Limits  Age
---------                   ----                                ------------  ----------  ---------------  -------------  ---
default                     frontend-deploy-6df94c895b-gpspl    0 (0%)        0 (0%)      0 (0%)           0 (0%)         2m25s
default                     frontend-deploy-6df94c895b-rlr4v    0 (0%)        0 (0%)      0 (0%)           0 (0%)         2m25s
default                     frontend-deploy-6df94c895b-tmkjp    0 (0%)        0 (0%)      0 (0%)           0 (0%)         2m25s
kube-system                 kindnet-snjtk                       100m (0%)     100m (0%)   50Mi (1%)        50Mi (1%)      61m
kube-system                 kube-proxy-zljdz                    0 (0%)        0 (0%)      0 (0%)           0 (0%)         61m
```

Table for Resource requests and Limits can be observed:

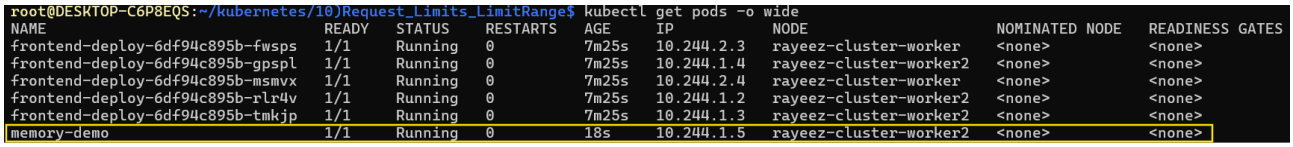
→ Let apply the following memory.yaml which similates the memory load on the container with 190Mi:

```
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo
spec:
  containers:
```

```
    - name: memory-demo-ctr
      image: polinux/stress
      resources:
        requests:
          memory: "100Mi"
        limits:
          memory: "200Mi"
      command: ["stress"]
      args: ["--vm", "1", "--vm-bytes", "190M",
"--vm-hang", "1"]
```

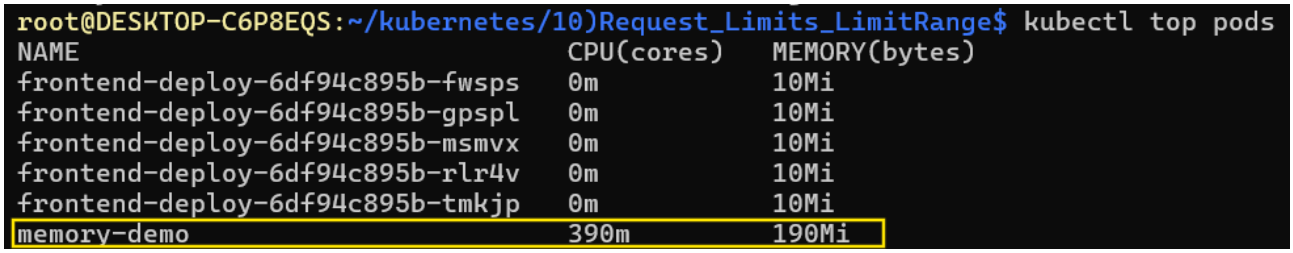
Apply;

```
kubectl apply -f memory.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl get pods -o wide
NAME                                 READY   STATUS    RESTARTS   AGE     IP            NODE                      NOMINATED NODE   READINESS GATES
frontend-deploy-6df94c895b-fwsps     1/1     Running   0          7m25s   10.244.2.3    rayeez-cluster-worker     <none>           <none>
frontend-deploy-6df94c895b-gpspl     1/1     Running   0          7m25s   10.244.1.4    rayeez-cluster-worker2    <none>           <none>
frontend-deploy-6df94c895b-msmvx     1/1     Running   0          7m25s   10.244.2.4    rayeez-cluster-worker     <none>           <none>
frontend-deploy-6df94c895b-rlr4v     1/1     Running   0          7m25s   10.244.1.2    rayeez-cluster-worker2    <none>           <none>
frontend-deploy-6df94c895b-tmkjp     1/1     Running   0          7m25s   10.244.1.3    rayeez-cluster-worker2    <none>           <none>
memory-demo                          1/1     Running   0          18s     10.244.1.5    rayeez-cluster-worker2    <none>           <none>
```

```
kubectl top pods
```

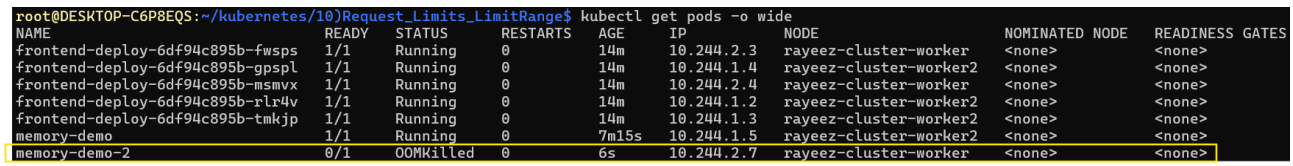


```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl top pods
NAME                                 CPU(cores)   MEMORY(bytes)
frontend-deploy-6df94c895b-fwsps     0m           10Mi
frontend-deploy-6df94c895b-gpspl     0m           10Mi
frontend-deploy-6df94c895b-msmvx     0m           10Mi
frontend-deploy-6df94c895b-rlr4v     0m           10Mi
frontend-deploy-6df94c895b-tmkjp     0m           10Mi
memory-demo                          390m         190Mi
```

→ Let's change this argument from **190Mi** to **250Mi**, which exceeds the container's defined limit of **200Mi** and apply a new pod.

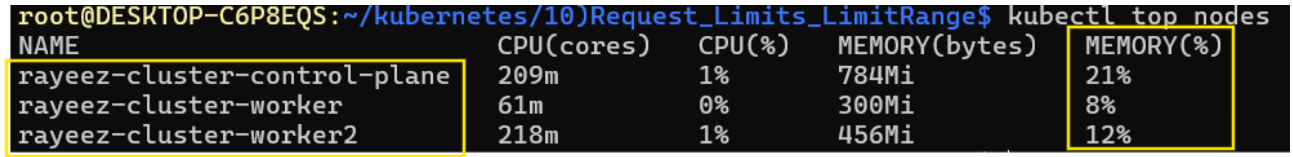
Apply;

```
kubectl apply -f memory.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl get pods -o wide
NAME                                 READY   STATUS      RESTARTS   AGE     IP            NODE                      NOMINATED NODE   READINESS GATES
frontend-deploy-6df94c895b-fwsps     1/1     Running     0          14m     10.244.2.3    rayeez-cluster-worker     <none>           <none>
frontend-deploy-6df94c895b-gpspl     1/1     Running     0          14m     10.244.1.4    rayeez-cluster-worker2    <none>           <none>
frontend-deploy-6df94c895b-msmvx     1/1     Running     0          14m     10.244.2.4    rayeez-cluster-worker     <none>           <none>
frontend-deploy-6df94c895b-rlr4v     1/1     Running     0          14m     10.244.1.2    rayeez-cluster-worker2    <none>           <none>
frontend-deploy-6df94c895b-tmkjp     1/1     Running     0          14m     10.244.1.3    rayeez-cluster-worker2    <none>           <none>
memory-demo                          1/1     Running     0          7m15s   10.244.1.5    rayeez-cluster-worker2    <none>           <none>
memory-demo-2                        0/1     OOMKilled   0          6s      10.244.2.7    rayeez-cluster-worker     <none>           <none>
```

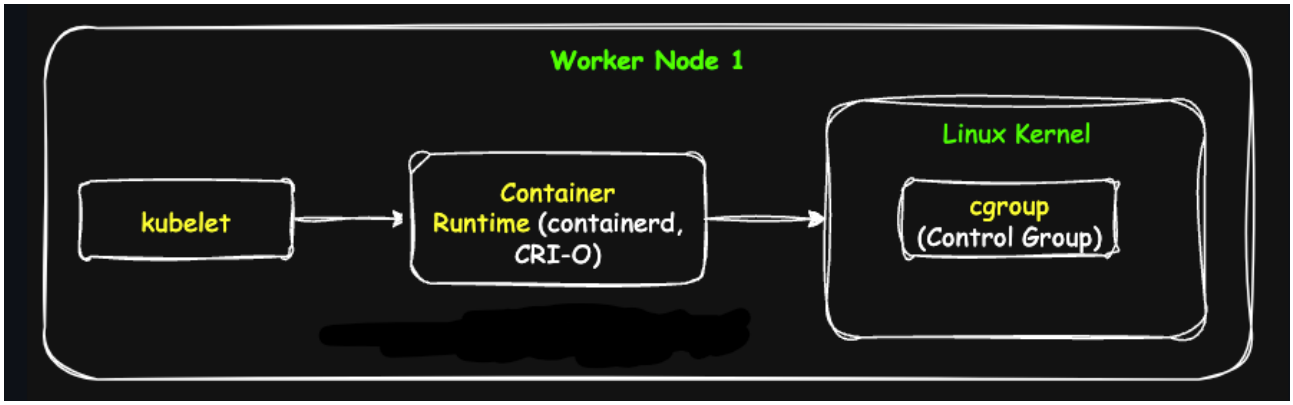
Pod Status is **OOMKilled**. But Nodes have enough resources available.



```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl top nodes
NAME                           CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
rayeez-cluster-control-plane   209m         1%       784Mi           21%
rayeez-cluster-worker          61m          0%       300Mi           8%
rayeez-cluster-worker2         218m         1%       456Mi           12%
```

It is clear that there is no **Memory Pressure**. Then Why pod is getting **OOMKilled**?

→ Concept of Control Groups(A Linux Concept):



1. Control groups essentially create resource boundaries, ensuring that each process is allowed to use only a defined amount of resources—such as memory, CPU, and I/O—without affecting others.
2. If a pod attempts to consume more resources than allowed, the **kernel** will terminate the process.
3. When the container runtime creates and starts a container inside a pod, it also sets up a corresponding **control group** for that container.
4. Each running pod corresponds to a process managed by a control group. If the **cgroup** detects that a pod is consuming more memory than what is defined in its resource specification, it terminates that process. This results in the pod entering the **OOMKilled** (Out of Memory Killed) state.

Lets create a Pod which similates the CPU load of 1000m which beyond the limit specified(900m):

```
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo
spec:
  containers:
    - name: cpu-container
```

```
      image: vish/stress
      resources:
        requests:
          cpu: "500m"
        limits:
          cpu: "900m"
      args:
        - -cpus
        - "1"
```

Apply;

```
kubectl apply -f cpu.yaml
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
cpu-demo                            899m         1Mi
frontend-deploy-6df94c895b-fwsps    0m           10Mi
frontend-deploy-6df94c895b-gpspl    0m           10Mi
frontend-deploy-6df94c895b-msmvx    0m           10Mi
frontend-deploy-6df94c895b-rlr4v    0m           10Mi
frontend-deploy-6df94c895b-tmkjp    0m           10Mi
memory-demo                         179m         190Mi
```

CPU Utilization is not going beyond **900m**.

The **kernel throttles** CPU usage when a container attempts to consume more CPU resources than the limits defined for it.

```
kubectl describe nodes rayeez-cluster-worker2
```

```
Namespace     Name                                CPU Requests  CPU Limits  Memory Requests  Memory Limits  Age
---------     ----                                ------------  ----------  ---------------  -------------  ---
default       cpu-demo                            500m (4%)     900m (7%)   0 (0%)           0 (0%)         6m27s
default       frontend-deploy-6df94c895b-gpspl    0 (0%)        0 (0%)      0 (0%)           0 (0%)         40m
default       frontend-deploy-6df94c895b-rlr4v    0 (0%)        0 (0%)      0 (0%)           0 (0%)         40m
default       frontend-deploy-6df94c895b-tmkjp    0 (0%)        0 (0%)      0 (0%)           0 (0%)         40m
default       memory-demo                         0 (0%)        0 (0%)      100Mi (2%)       200Mi (5%)     33m
kube-system   kindnet-snjtk                       100m (0%)     100m (0%)   50Mi (1%)        50Mi (1%)      99m
kube-system   kube-proxy-zljdz                    0 (0%)        0 (0%)      0 (0%)           0 (0%)         99m
```

Lets Explore **LimitRange** Object of K8s:

→ **LimitRange**: Setting Default Requests and Limits

A LimitRange enforces default CPU and memory requests/limits in a **namespace**, automatically applying them when containers don't specify their own.

Let apply the following LimitRange yaml:

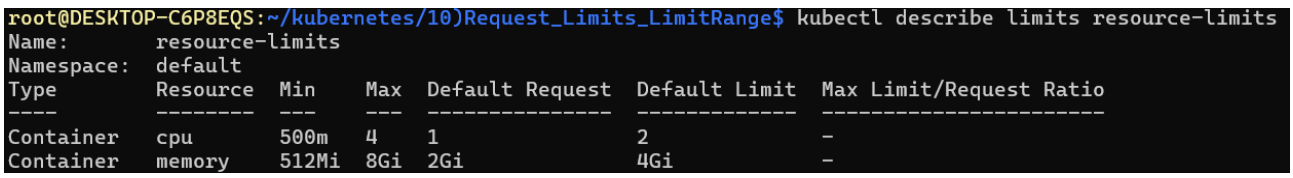
```
apiVersion: v1
kind: LimitRange
metadata:
  name: resource-limits
  namespace: default
spec:
  limits:
    - type: Container
      default:
        cpu: "2" # Default CPU limit
        memory: "4Gi" # Default memory limit
      defaultRequest:
        cpu: "1" # Default CPU request
        memory: "2Gi" # Default memory request
      max:
        cpu: "4" # Maximum CPU a container can request
        memory: "8Gi" # Maximum memory a container can request
      min:
        cpu: "500m" # Minimum CPU a container must request
        memory: "512Mi" # Minimum memory a container must request
```

Apply;

```
kubectl apply -f limitrange.yaml
```

Verify;

```
kubectl describe limits resource-limits
```



Lets create a pod imperatively and check its specifications:

```
kubectl run test-pod --image=nginx
kubectl describe pods test-pod
```

```
Ready:          True
Restart Count:  0
Limits:
  cpu:     2
  memory:  4Gi
Requests:
  cpu:        1
  memory:     2Gi
Environment:  <none>
```

Requests and Limits are set for this pod because of LimitRange applied in default namespece

→ Lets try to create a pod less than 500m CPU:

```
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo-2
spec:
  containers:
    - name: cpu-container
      image: vish/stress
      resources:
        requests:
          cpu: "400m"
        limits:
          cpu: "900m"
      args:
        - -cpus
        - "1"
```

Apply;

```
kubectl apply -f cpu.yaml
```

```
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$ kubectl apply -f cpu.yaml
Error from server (Forbidden): error when creating "cpu.yaml": pods "cpu-demo-2" is forbidden: minimum cpu usage per Container is 500m, but request is 400m
root@DESKTOP-C6P8EQS:~/kubernetes/10)Request_Limits_LimitRange$
```

It gives an error: minimum cpu usage per Container is **500m**, but request is **400m**.

This happens because the LimitRange enforces a minimum CPU request of 500m for every container.