

# Resize CPU and Memory Resources assigned to Containers

**FEATURE STATE:** [Kubernetes v1.27](#) [\[alpha\]](#)

This page assumes that you are familiar with [Quality of Service](#) for Kubernetes Pods.

This page shows how to resize CPU and memory resources assigned to containers of a running pod without restarting the pod or its containers. A Kubernetes node allocates resources for a pod based on its `requests`, and restricts the pod's resource usage based on the `limits` specified in the pod's containers.

For in-place resize of pod resources:

- Container's resource `requests` and `limits` are *mutable* for CPU and memory resources.
- `allocatedResources` field in `containerStatuses` of the Pod's status reflects the resources allocated to the pod's containers.
- `resources` field in `containerStatuses` of the Pod's status reflects the actual resource `requests` and `limits` that are configured on the running containers as reported by the container runtime.
- `resize` field in the Pod's status shows the status of the last requested pending resize. It can have the following values:
  - `Proposed` : This value indicates an acknowledgement of the requested resize and that the request was validated and recorded.
  - `InProgress` : This value indicates that the node has accepted the resize request and is in the process of applying it to the pod's containers.
  - `Deferred` : This value means that the requested resize cannot be granted at this time, and the node will keep retrying. The resize may be granted when other pods leave and free up node resources.
  - `Infeasible` : is a signal that the node cannot accommodate the requested resize. This can happen if the requested resize exceeds the maximum resources the node can ever allocate for a pod.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Killercode](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version 1.27. To check the version, enter `kubectl version`.

## Container Resize Policies

Resize policies allow for a more fine-grained control over how pod's containers are resized for CPU and memory resources. For example, the container's application may be able to handle CPU resources resized without being restarted, but resizing memory may require that the application hence the containers be restarted.

To enable this, the Container specification allows users to specify a `resizePolicy`. The following restart policies can be specified for resizing CPU and memory:

- `NotRequired` : Resize the container's resources while it is running.
- `RestartContainer` : Restart the container and apply new resources upon restart.

If `resizePolicy[*].restartPolicy` is not specified, it defaults to `NotRequired`.

**Note:** If the Pod's `restartPolicy` is `Never`, container's resize restart policy must be set to `NotRequired` for all Containers in the Pod.


Below example shows a Pod whose Container's CPU can be resized without restart, but resizing memory requires the container to be restarted.

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo-5
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr-5
    image: nginx
    resizePolicy:
      - resourceName: cpu
        restartPolicy: NotRequired
      - resourceName: memory
        restartPolicy: RestartContainer
    resources:
      limits:
        memory: "200Mi"
        cpu: "700m"
      requests:
        memory: "200Mi"
        cpu: "700m"
```

**Note:** In the above example, if desired requests or limits for both CPU *and* memory have changed, the container will be restarted in order to resize its memory.

## Create a pod with resource requests and limits

You can create a Guaranteed or Burstable [Quality of Service](#) class pod by specifying requests and/or limits for a pod's containers. Consider the following manifest for a Pod that has one Container.

pods/qos/qos-pod-5.yaml 

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo-5
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr-5
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "700m"
      requests:
        memory: "200Mi"
        cpu: "700m"
```

Create the pod in the `qos-example` namespace:

```
kubectl create namespace qos-example
kubectl create -f https://k8s.io/examples/pods/qos/qos-pod-5.yaml
```

This pod is classified as a Guaranteed QoS class requesting 700m CPU and 200Mi memory.

View detailed information about the pod:

```
kubectl get pod qos-demo-5 --output=yaml --namespace=qos-example
```

Also notice that the values of `resizePolicy[*].restartPolicy` defaulted to `NotRequired`, indicating that CPU and memory can be resized while container is running.

```
spec:
  containers:
    ...
    resizePolicy:
    - resourceName: cpu
      restartPolicy: NotRequired
    - resourceName: memory
      restartPolicy: NotRequired
    resources:
      limits:
        cpu: 700m
        memory: 200Mi
      requests:
        cpu: 700m
        memory: 200Mi
    ...
  containerStatuses:
  ...
  name: qos-demo-ctr-5
  ready: true
  ...
  allocatedResources:
    cpu: 700m
    memory: 200Mi
  resources:
    limits:
      cpu: 700m
      memory: 200Mi
    requests:
      cpu: 700m
      memory: 200Mi
    restartCount: 0
    started: true
  ...
  qosClass: Guaranteed
```

## Updating the pod's resources

Let's say the CPU requirements have increased, and 0.8 CPU is now desired. This is typically determined, and may be programmatically applied, by an entity such as [VerticalPodAutoscaler](#) (VPA).

**Note:** While you can change a Pod's requests and limits to express new desired resources, you cannot change the QoS class in which the Pod was created.

Now, patch the Pod's Container with CPU requests & limits both set to `800m`:

```
kubectl -n qos-example patch pod qos-demo-5 --patch '{"spec":{"containers":[{"name":"qos-demo-ctr-5", "resources":{"
```

Query the Pod's detailed information after the Pod has been patched.

```
kubectl get pod qos-demo-5 --output=yaml --namespace=qos-example
```

The Pod's spec below reflects the updated CPU requests and limits.

```
spec:
  containers:
    ...
    resources:
      limits:
        cpu: 800m
        memory: 200Mi
      requests:
        cpu: 800m
        memory: 200Mi
    ...
  containerStatuses:
    ...
    allocatedResources:
      cpu: 800m
      memory: 200Mi
    resources:
      limits:
        cpu: 800m
        memory: 200Mi
      requests:
        cpu: 800m
        memory: 200Mi
    restartCount: 0
    started: true
```

Observe that the `allocatedResources` values have been updated to reflect the new desired CPU requests. This indicates that node was able to accommodate the increased CPU resource needs.

In the Container's status, updated CPU resource values shows that new CPU resources have been applied. The Container's `restartCount` remains unchanged, indicating that container's CPU resources were resized without restarting the container.

## Clean up

Delete your namespace:

```
kubectl delete namespace qos-example
```

## What's next

For application developers

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)

## For cluster administrators

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)

## Feedback

Was this page helpful?

Yes

No

---

Last modified August 24, 2023 at 6:38 PM PST: [Use code\\_sample shortcode instead of code shortcode \(e8b136c3b3\)](#)