**CPU Management in Kubernetes**

**Manage CPU Limits for PODs**

Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

**kubectl create namespace cpu-example**

Specify a CPU request and a CPU limit

To specify a CPU request for a Container, include the **resources:requests** field in the Container’s resource manifest. To specify a CPU limit, include **resources:limits**.

In this exercise, you create a Pod that has one Container. The Container has a CPU request of 0.5 cpu and a CPU limit of 1 cpu. Here’s the configuration file for the Pod:

| [**cpu-request-limit.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/configure-pod-container/cpu-request-limit.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: cpu-demo**  **spec:**  **containers:**  **- name: cpu-demo-ctr**  **image: vish/stress**  **resources:**  **limits:**  **cpu: "1"**  **requests:**  **cpu: "0.5"**  **args:**  **- -cpus**  **- "2"** |
|  |

In the configuration file, the **args** section provides arguments for the Container when it starts. The **-cpus "2"**argument tells the Container to attempt to use 2 cpus.

Create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/configure-pod-container/cpu-request-limit.yaml --namespace=cpu-example**

Verify that the Pod’s Container is running:

**kubectl get pod cpu-demo --namespace=cpu-example**

View detailed information about the Pod:

**kubectl get pod cpu-demo --output=yaml --namespace=cpu-example**

The output shows that the one Container in the Pod has a CPU request of 500 millicpu and a CPU limit of 1 cpu.

**resources:**

**limits:**

**cpu: "1"**

**requests:**

**cpu: 500m**

Start a proxy so that you can call the heapster service:

**kubectl proxy**

In another command window, get the CPU usage rate from the heapster service:

**curl http://localhost:8001/api/v1/proxy/namespaces/kube-system/services/heapster/api/v1/model/namespaces/cpu-example/pods/cpu-demo/metrics/cpu/usage\_rate**

The output shows that the Pod is using 974 millicpu, which is just a bit less than the limit of 1 cpu specified in the Pod’s configuration file.

**{**

**"timestamp": "2017-06-22T18:48:00Z",**

**"value": 974**

**}**

Recall that by setting **-cpu "2"**, you configured the Container to attempt to use 2 cpus. But the Container is only being allowed to use about 1 cpu. The Container’s CPU use is being throttled, because the Container is attempting to use more CPU resources than its limit.

**Note:** There’s another possible explanation for the CPU throttling. The Node might not have enough CPU resources available. Recall that the prerequisites for this exercise require that each of your Nodes has at least 1 cpu. If your Container is running on a Node that has only 1 cpu, the Container cannot use more than 1 cpu regardless of the CPU limit specified for the Container.

CPU units

The CPU resource is measured in *cpu* units. One cpu, in Kubernetes, is equivalent to:

* 1 AWS vCPU
* 1 GCP Core
* 1 Azure vCore
* 1 Hyperthread on a bare-metal Intel processor with Hyperthreading

Fractional values are allowed. A Container that requests 0.5 cpu is guaranteed half as much CPU as a Container that requests 1 cpu. You can use the suffix m to mean milli. For example 100m cpu, 100 millicpu, and 0.1 cpu are all the same. Precision finer than 1m is not allowed.

CPU is always requested as an absolute quantity, never as a relative quantity; 0.1 is the same amount of CPU on a single-core, dual-core, or 48-core machine.

Delete your Pod:

**kubectl delete pod cpu-demo --namespace=cpu-example**

Specify a CPU request that is too big for your Nodes

CPU requests and limits are associated with Containers, but it is useful to think of a Pod as having a CPU request and limit. The CPU request for a Pod is the sum of the CPU requests for all the Containers in the Pod. Likewise, the CPU limit for a Pod is the sum of the CPU limits for all the Containers in the Pod.

Pod scheduling is based on requests. A Pod is scheduled to run on a Node only if the Node has enough CPU resources available to satisfy the Pod’s CPU request.

In this exercise, you create a Pod that has a CPU request so big that it exceeds the capacity of any Node in your cluster. Here is the configuration file for a Pod that has one Container. The Container requests 100 cpu, which is likely to exceed the capacity of any Node in your cluster.

| [**cpu-request-limit-2.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/configure-pod-container/cpu-request-limit-2.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: cpu-demo-2**  **spec:**  **containers:**  **- name: cpu-demo-ctr-2**  **image: vish/stress**  **resources:**  **limits:**  **cpu: "100"**  **requests:**  **cpu: "100"**  **args:**  **- -cpus**  **- "2"** |

Create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/configure-pod-container/cpu-request-limit-2.yaml --namespace=cpu-example**

View the Pod’s status:

**kubectl get pod cpu-demo-2 --namespace=cpu-example**

The output shows that the Pod’s status is Pending. That is, the Pod has not been scheduled to run on any Node, and it will remain in the Pending state indefinitely:

**kubectl get pod cpu-demo-2 --namespace=cpu-example**

**NAME READY STATUS RESTARTS AGE**

**cpu-demo-2 0/1 Pending 0 7m**

View detailed information about the Pod, including events:

**kubectl describe pod cpu-demo-2 --namespace=cpu-example**

The output shows that the Container cannot be scheduled because of insufficient CPU resources on the Nodes:

**Events:**

**Reason Message**

**------ -------**

**FailedScheduling No nodes are available that match all of the following predicates:: Insufficient cpu (3).**

Delete your Pod:

**kubectl delete pod cpu-demo-2 --namespace=cpu-example**

If you don’t specify a CPU limit

If you don’t specify a CPU limit for a Container, then one of these situations applies:

* The Container has no upper bound on the CPU resources it can use. The Container could use all of the CPU resources available on the Node where it is running.
* The Container is running in a namespace that has a default CPU limit, and the Container is automatically assigned the default limit. Cluster administrators can use a [LimitRange](https://kubernetes.io/docs/api-reference/v1.8/" \l "limitrange-v1-core/) to specify a default value for the CPU limit.

Motivation for CPU requests and limits

By configuring the CPU requests and limits of the Containers that run in your cluster, you can make efficient use of the CPU resources available on your cluster’s Nodes. By keeping a Pod’s CPU request low, you give the Pod a good chance of being scheduled. By having a CPU limit that is greater than the CPU request, you accomplish two things:

* The Pod can have bursts of activity where it makes use of CPU resources that happen to be available.
* The amount of CPU resources a Pod can use during a burst is limited to some reasonable amount.

Clean up

Delete your namespace:

**kubectl delete namespace cpu-example**

# **Configure Default CPU Requests and Limits for a Namespace**

Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

**kubectl create namespace default-cpu-example**

Create a LimitRange and a Pod

Here’s the configuration file for a LimitRange object. The configuration specifies a default CPU request and a default CPU limit.

| [**cpu-defaults.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-defaults.yaml) |
| --- |
| **apiVersion: v1**  **kind: LimitRange**  **metadata:**  **name: cpu-limit-range**  **spec:**  **limits:**  **- default:**  **cpu: 1**  **defaultRequest:**  **cpu: 0.5**  **type: Container** |

Create the LimitRange in the default-cpu-example namespace:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-defaults.yaml --namespace=default-cpu-example**

Now if a Container is created in the default-cpu-example namespace, and the Container does not specify its own values for CPU request and CPU limit, the Container is given a default CPU request of 0.5 and a default CPU limit of 1.

Here’s the configuration file for a Pod that has one Container. The Container does not specify a CPU request and limit.

| [**cpu-defaults-pod.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-defaults-pod.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: default-cpu-demo**  **spec:**  **containers:**  **- name: default-cpu-demo-ctr**  **image: nginx** |

Create the Pod.

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-defaults-pod.yaml --namespace=default-cpu-example**

View the Pod’s specification:

**kubectl get pod default-cpu-demo --output=yaml --namespace=default-cpu-example**

The output shows that the Pod’s Container has a CPU request of 500 millicpus and a CPU limit of 1 cpu. These are the default values specified by the LimitRange.

**containers:**

**- image: nginx**

**imagePullPolicy: Always**

**name: default-cpu-demo-ctr**

**resources:**

**limits:**

**cpu: "1"**

**requests:**

**cpu: 500m**

What if you specify a Container’s limit, but not its request?

Here’s the configuration file for a Pod that has one Container. The Container specifies a CPU limit, but not a request:

| [**cpu-defaults-pod-2.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-defaults-pod-2.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: default-cpu-demo-2**  **spec:**  **containers:**  **- name: default-cpu-demo-2-ctr**  **image: nginx**  **resources:**  **limits:**  **cpu: "1"** |

Create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-defaults-pod-2.yaml --namespace=default-cpu-example**

View the Pod specification:

**kubectl get pod default-cpu-demo-2 --output=yaml --namespace=default-cpu-example**

The output shows that the Container’s CPU request is set to match its CPU limit. Notice that the Container was not assigned the default CPU request value of 0.5 cpu.

**resources:**

**limits:**

**cpu: "1"**

**requests:**

**cpu: "1"**

What if you specify a Container’s request, but not its limit?

Here’s the configuration file for a Pod that has one Container. The Container specifies a CPU request, but not a limit:

| [**cpu-defaults-pod-3.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-defaults-pod-3.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: default-cpu-demo-3**  **spec:**  **containers:**  **- name: default-cpu-demo-3-ctr**  **image: nginx**  **resources:**  **requests:**  **cpu: "0.75"** |

Create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-defaults-pod-3.yaml --namespace=default-cpu-example**

View the Pod specification:

**kubectl get pod default-cpu-demo-3 --output=yaml --namespace=default-cpu-example**

The output shows that the Container’s CPU request is set to the value specified in the Container’s configuration file. The Container’s CPU limit is set to 1 cpu, which is the default CPU limit for the namespace.

**resources:**

**limits:**

**cpu: "1"**

**requests:**

**cpu: 750m**

Motivation for default CPU limits and requests

If your namespace has a [resource quota](https://kubernetes.io/docs/tasks/administer-cluster/cpu-default-namespace/), it is helpful to have a default value in place for CPU limit. Here are two of the restrictions that a resource quota imposes on a namespace:

* Every Container that runs in the namespace must have its own CPU limit.
* The total amount of CPU used by all Containers in the namespace must not exceed a specified limit.

If a Container does not specify its own CPU limit, it is given the default limit, and then it can be allowed to run in a namespace that is restricted by a quota.

# **Configure Minimum and Maximum CPU Constraints for a Namespace**

Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

**kubectl create namespace constraints-cpu-example**

Create a LimitRange and a Pod

Here’s the configuration file for a LimitRange:

| [**cpu-constraints.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-constraints.yaml) |
| --- |
| **apiVersion: v1**  **kind: LimitRange**  **metadata:**  **name: cpu-min-max-demo-lr**  **spec:**  **limits:**  **- max:**  **cpu: "800m"**  **min:**  **cpu: "200m"**  **type: Container** |

Create the LimitRange:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-constraints.yaml --namespace=constraints-cpu-example**

View detailed information about the LimitRange:

**kubectl get limitrange cpu-min-max-demo-lr --output=yaml --namespace=constraints-cpu-example**

The output shows the minimum and maximum CPU constraints as expected. But notice that even though you didn’t specify default values in the configuration file for the LimitRange, they were created automatically.

**limits:**

**- default:**

**cpu: 800m**

**defaultRequest:**

**cpu: 800m**

**max:**

**cpu: 800m**

**min:**

**cpu: 200m**

**type: Container**

Now whenever a Container is created in the constraints-cpu-example namespace, Kubernetes performs these steps:

* If the Container does not specify its own CPU request and limit, assign the default CPU request and limit to the Container.
* Verify that the Container specifies a CPU request that is greater than or equal to 200 millicpu.
* Verify that the Container specifies a CPU limit that is less than or equal to 800 millicpu.

Here’s the configuration file for a Pod that has one Container. The Container manifest specifies a CPU request of 500 millicpu and a CPU limit of 800 millicpu. These satisfy the minimum and maximum CPU constraints imposed by the LimitRange.

| [**cpu-constraints-pod.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-constraints-pod.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: constraints-cpu-demo**  **spec:**  **containers:**  **- name: constraints-cpu-demo-ctr**  **image: nginx**  **resources:**  **limits:**  **cpu: "800m"**  **requests:**  **cpu: "500m"** |

Create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-constraints-pod.yaml --namespace=constraints-cpu-example**

Verify that the Pod’s Container is running:

**kubectl get pod constraints-cpu-demo --namespace=constraints-cpu-example**

View detailed information about the Pod:

**kubectl get pod constraints-cpu-demo --output=yaml --namespace=constraints-cpu-example**

The output shows that the Container has a CPU request of 500 millicpu and CPU limit of 800 millicpu. These satisfy the constraints imposed by the LimitRange.

**resources:**

**limits:**

**cpu: 800m**

**requests:**

**cpu: 500m**

Delete the Pod

**kubectl delete pod constraints-cpu-demo --namespace=constraints-cpu-example**

Attempt to create a Pod that exceeds the maximum CPU constraint

Here’s the configuration file for a Pod that has one Container. The Container specifies a CPU request of 500 millicpu and a cpu limit of 1.5 cpu.

| [**cpu-constraints-pod-2.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-constraints-pod-2.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: constraints-cpu-demo-2**  **spec:**  **containers:**  **- name: constraints-cpu-demo-2-ctr**  **image: nginx**  **resources:**  **limits:**  **cpu: "1.5"**  **requests:**  **cpu: "500m"** |

Attempt to create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-constraints-pod-2.yaml --namespace=constraints-cpu-example**

The output shows that the Pod does not get created, because the Container specifies a CPU limit that is too large:

**Error from server (Forbidden): error when creating "docs/tasks/administer-cluster/cpu-constraints-pod-2.yaml":**

**pods "constraints-cpu-demo-2" is forbidden: maximum cpu usage per Container is 800m, but limit is 1500m.**

Attempt to create a Pod that does not meet the minimum CPU request

Here’s the configuration file for a Pod that has one Container. The Container specifies a CPU request of 100 millicpu and a CPU limit of 800 millicpu.

| [**cpu-constraints-pod-3.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-constraints-pod-3.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: constraints-cpu-demo-4**  **spec:**  **containers:**  **- name: constraints-cpu-demo-4-ctr**  **image: nginx**  **resources:**  **limits:**  **cpu: "800m"**  **requests:**  **cpu: "100m"** |

Attempt to create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-constraints-pod-3.yaml --namespace=constraints-cpu-example**

The output shows that the Pod does not get created, because the Container specifies a CPU request that is too small:

**Error from server (Forbidden): error when creating "docs/tasks/administer-cluster/cpu-constraints-pod-3.yaml":**

**pods "constraints-cpu-demo-4" is forbidden: minimum cpu usage per Container is 200m, but request is 100m.**

Create a Pod that does not specify any CPU request or limit

Here’s the configuration file for a Pod that has one Container. The Container does not specify a CPU request, and it does not specify a CPU limit.

| [**cpu-constraints-pod-4.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/docs/tasks/administer-cluster/cpu-constraints-pod-4.yaml) |
| --- |
| **apiVersion: v1**  **kind: Pod**  **metadata:**  **name: constraints-cpu-demo-4**  **spec:**  **containers:**  **- name: constraints-cpu-demo-4-ctr**  **image: vish/stress** |

Create the Pod:

**kubectl create -f https://k8s.io/docs/tasks/administer-cluster/cpu-constraints-pod-4.yaml --namespace=constraints-cpu-example**

View detailed information about the Pod:

**kubectl get pod constraints-cpu-demo-4 --namespace=constraints-cpu-example --output=yaml**

The output shows that the Pod’s Container has a CPU request of 800 millicpu and a CPU limit of 800 millicpu. How did the Container get those values?

**resources:**

**limits:**

**cpu: 800m**

**requests:**

**cpu: 800m**

Because your Container did not specify its own CPU request and limit, it was given the [default CPU request and limit](https://kubernetes.io/docs/tasks/administer-cluster/cpu-default-namespace/) from the LimitRange.

At this point, your Container might be running or it might not be running. Recall that a prerequisite for this task is that your Nodes have at least 1 CPU. If each of your Nodes has only 1 CPU, then there might not be enough allocatable CPU on any Node to accommodate a request of 800 millicpu. If you happen to be using Nodes with 2 CPU, then you probably have enough CPU to accommodate the 800 millicpu request.

Delete your Pod:

**kubectl delete pod constraints-cpu-demo-4 --namespace=constraints-cpu-example**

Enforcement of minimum and maximum CPU constraints

The maximum and minimum CPU constraints imposed on a namespace by a LimitRange are enforced only when a Pod is created or updated. If you change the LimitRange, it does not affect Pods that were created previously.

Motivation for minimum and maximum CPU constraints

As a cluster administrator, you might want to impose restrictions on the CPU resources that Pods can use. For example:

* Each Node in a cluster has 2 CPU. You do not want to accept any Pod that requests more than 2 CPU, because no Node in the cluster can support the request.
* A cluster is shared by your production and development departments. You want to allow production workloads to consume up to 3 CPU, but you want development workloads to be limited to 1 CPU. You create separate namespaces for production and development, and you apply CPU constraints to each namespace.

Clean up

Delete your namespace:

**kubectl delete namespace constraints-cpu-example**