Resource constraints

Table of contents

By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. Docker provides ways to control how much memory, or CPU a container can use, setting runtime configuration flags of the `docker run` command. This section provides details on when you should set such limits and the possible implications of setting them.

Many of these features require your kernel to support Linux capabilities. To check for support, you can use the `docker info` command. If a capability is disabled in your kernel, you may see a warning at the end of the output like the following:

```
WARNING: No swap limit support
```

Consult your operating system's documentation for enabling them. See also the Docker Engine troubleshooting guide for more information.

Memory

Understand the risks of running out of memory

It's important not to allow a running container to consume too much of the host machine's memory. On Linux hosts, if the kernel detects that there isn't enough memory to perform important system functions, it throws an `OOME`, or `Out Of Memory Exception`, and starts killing processes to free up memory. Any process is subject to killing, including Docker and other important applications. This can effectively bring the entire system down if the wrong process is killed.

Docker attempts to mitigate these risks by adjusting the OOM priority on the Docker daemon so that it's less likely to be killed than other processes on the system. The OOM priority on containers isn't adjusted. This makes it more likely for an individual container to be killed than for the Docker daemon or other system processes to be killed. You shouldn't try to circumvent these safeguards by manually setting `--oom-score-adj` to an extreme negative number on the daemon or a container, or by setting `--oom-kill-disable` on a container.

For more information about the Linux kernel's OOM management, see Out of Memory Management.

You can mitigate the risk of system instability due to OOME by:

- Perform tests to understand the memory requirements of your application before placing it into production.
- Ensure that your application runs only on hosts with adequate resources.
- Limit the amount of memory your container can use, as described below.
- Be mindful when configuring swap on your Docker hosts. Swap is slower than memory but can provide a buffer against running out of system memory.

- Consider converting your container to a service, and using service-level constraints and node labels to ensure that the application runs only on hosts with enough memory

Limit a container's access to memory

Docker can enforce hard or soft memory limits.

- Hard limits lets the container use no more than a fixed amount of memory.
- Soft limits lets the container use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine.

Some of these options have different effects when used alone or when more than one option is set.

Most of these options take a positive integer, followed by a suffix of `b`, `k`, `m`, `g`, to indicate bytes, kilobytes, megabytes, or gigabytes.

Search Ask AI

`-m` or `--memory=`	The maximum amount of memory the container can use. If you set this option, the minimum allowed value is `6m` (6 megabytes). That is, you must set the value to at least 6 megabytes.
`--memory-swap`*	The amount of memory this container is allowed to swap to disk. See `--memory-swap` details.
`--memory-swappiness`	By default, the host kernel can swap out a percentage of anonymous pages used by a container. You can set `--memory-swappiness` to a value between 0 and 100, to tune this percentage. See `--memory-swappiness` details.
`--memory-reservation`	Allows you to specify a soft limit smaller than `--memory` which is activated when Docker detects contention or low memory on the host machine. If you use `--memory-reservation`, it must be set lower than `--memory` for it to take precedence. Because it is a soft limit, it doesn't guarantee that the container doesn't exceed the limit.
`--kernel-memory`	The maximum amount of kernel memory the container can use. The minimum allowed value is `6m`. Because kernel memory can't be swapped out, a container which is starved of kernel memory may block host machine

Option	Description
	resources, which can have side effects on the host machine and on other containers. See `--kernel-memory` details.
`--oom-kill-disable`	By default, if an out-of-memory (OOM) error occurs, the kernel kills processes in a container. To change this behavior, use the `--oom-kill-disable` option. Only disable the OOM killer on containers where you have also set the `-m/--memory` option. If the `-m` flag isn't set, the host can run out of memory and the kernel may need to kill the host system's processes to free memory.

For more information about cgroups and memory in general, see the documentation for Memory Resource Controller.

`--memory-swap` details

`--memory-swap` is a modifier flag that only has meaning if `--memory` is also set. Using swap allows the container to write excess memory requirements to disk when the container has exhausted all the RAM that's available to it. There is a performance penalty for applications that swap memory to disk often.

Its setting can have complicated effects:

- If `--memory-swap` is set to a positive integer, then both `--memory` and `--memory-swap` must be set. `--memory-swap` represents the total amount of memory and swap that can be used, and `--memory` controls the amount used by non-swap memory. So if `--memory="300m"` and `--memory-swap="1g"`, the container can use 300m of memory and 700m (`1g - 300m`) swap.
- If `--memory-swap` is set to `0`, the setting is ignored, and the value is treated as unset.
- If `--memory-swap` is set to the same value as `--memory`, and `--memory` is set to a positive integer, **the container doesn't have access to swap**. See Prevent a container from using swap.
- If `--memory-swap` is unset, and `--memory` is set, the container can use as much swap as the `--memory` setting, if the host container has swap memory configured. For instance, if

`--memory="300m"` and `--memory-swap` is not set, the container can use 600m in total of memory and swap.

- If `--memory-swap` is explicitly set to `-1`, the container is allowed to use unlimited swap, up to the amount available on the host system.
- Inside the container, tools like `free` report the host's available swap, not what's available inside the container. Don't rely on the output of `free` or similar tools to determine whether swap is present.

Prevent a container from using swap

If `--memory` and `--memory-swap` are set to the same value, this prevents containers from using any swap. This is because `--memory-swap` is the amount of combined memory and swap that can be used, while `--memory` is only the amount of physical memory that can be used.

`--memory-swappiness` details

- A value of 0 turns off anonymous page swapping.
- A value of 100 sets all anonymous pages as swappable.
- By default, if you don't set `--memory-swappiness`, the value is inherited from the host machine.

`--kernel-memory` details

Kernel memory limits are expressed in terms of the overall memory allocated to a container. Consider the following scenarios:

- **Unlimited memory, unlimited kernel memory**: This is the default behavior.
- **Unlimited memory, limited kernel memory**: This is appropriate when the amount of memory needed by all cgroups is greater than the amount of memory that actually exists on the host machine. You can configure the kernel memory to never go over what's available on the host machine, and containers which need more memory need to wait for it.
- **Limited memory, unlimited kernel memory**: The overall memory is limited, but the kernel memory isn't.

- **Limited memory, limited kernel memory**: Limiting both user and kernel memory can be useful for debugging memory-related problems. If a container is using an unexpected amount of either type of memory, it runs out of memory without affecting other containers or the host machine. Within this setting, if the kernel memory limit is lower than the user memory limit, running out of kernel memory causes the container to experience an OOM error. If the kernel memory limit is higher than the user memory limit, the kernel limit doesn't cause the container to experience an OOM.

When you enable kernel memory limits, the host machine tracks "high water mark" statistics on a per-process basis, so you can track which processes (in this case, containers) are using excess memory. This can be seen per process by viewing `/proc/<PID>/status` on the host machine.

CPU

By default, each container's access to the host machine's CPU cycles is unlimited. You can set various constraints to limit a given container's access to the host machine's CPU cycles. Most users use and configure the default CFS scheduler. You can also configure the real-time scheduler.

Configure the default CFS scheduler

The CFS is the Linux kernel CPU scheduler for normal Linux processes. Several runtime flags let you configure the amount of access to CPU resources your container has. When you use these settings, Docker modifies the settings for the container's cgroup on the host machine.

Option	Description
`--cpus=<value>`	Specify how much of the available CPU resources a container can use. For instance, if the host machine has two CPUs and you set `--cpus="1.5"`, the container is guaranteed at most one and a half of the CPUs. This is the equivalent of setting `--cpu-period="100000"` and `--cpu-quota="150000"`.
`--cpu-period=<value>`	Specify the CPU CFS scheduler period, which is used alongside `--cpu-quota`. Defaults to 100000 microseconds (100 milliseconds). Most users don't

Option	Description
	change this from the default. For most use-cases, `--cpus` is a more convenient alternative.
`--cpu-quota=<value>`	Impose a CPU CFS quota on the container. The number of microseconds per `--cpu-period` that the container is limited to before throttled. As such acting as the effective ceiling. For most use-cases, `--cpus` is a more convenient alternative.
`--cpuset-cpus`	Limit the specific CPUs or cores a container can use. A comma-separated list or hyphen-separated range of CPUs a container can use, if you have more than one CPU. The first CPU is numbered 0. A valid value might be `0-3` (to use the first, second, third, and fourth CPU) or `1,3` (to use the second and fourth CPU).
`--cpu-shares`	Set this flag to a value greater or less than the default of 1024 to increase or reduce the container's weight, and give it access to a greater or lesser proportion of the host machine's CPU cycles. This is only enforced when CPU cycles are constrained. When plenty of CPU cycles are available, all containers use as much CPU as they need. In that way, this is a soft limit. `--cpu-shares` doesn't prevent containers from being scheduled in Swarm mode. It prioritizes container CPU resources for the available CPU cycles. It doesn't guarantee or reserve any specific CPU access.

If you have 1 CPU, each of the following commands guarantees the container at most 50% of the CPU every second.

```
$ docker run -it --cpus=".5" ubuntu /bin/bash
```

Which is the equivalent to manually specifying `--cpu-period` and `--cpu-quota`;

```
$ docker run -it --cpu-period=100000 --cpu-quota=50000 ubuntu /bin/bash
```

Configure the real-time scheduler

You can configure your container to use the real-time scheduler, for tasks which can't use the CFS scheduler. You need to make sure the host machine's kernel is configured correctly before you can configure the Docker daemon or configure individual containers.

> **Warning**
>
> CPU scheduling and prioritization are advanced kernel-level features. Most users don't need to change these values from their defaults. Setting these values incorrectly can cause your host system to become unstable or unusable.

Configure the host machine's kernel

Verify that `CONFIG_RT_GROUP_SCHED` is enabled in the Linux kernel by running `zcat /proc/config.gz | grep CONFIG_RT_GROUP_SCHED` or by checking for the existence of the file `/sys/fs/cgroup/cpu.rt_runtime_us`. For guidance on configuring the kernel real-time scheduler, consult the documentation for your operating system.

Configure the Docker daemon

To run containers using the real-time scheduler, run the Docker daemon with the `--cpu-rt-runtime` flag set to the maximum number of microseconds reserved for real-time tasks per runtime period. For instance, with the default period of 1000000 microseconds (1 second), setting `--cpu-rt-runtime=950000` ensures that containers using the real-time scheduler can run for 950000 microseconds for every 1000000-microsecond period, leaving at least 50000 microseconds available for non-real-time tasks. To make this configuration permanent on systems which use `systemd`, create a systemd unit file for the `docker` service. For an example, see the instruction on how to configure the daemon to use a proxy with a systemd unit file.

Configure individual containers

You can pass several flags to control a container's CPU priority when you start the container using `docker run`. Consult your operating system's documentation or the `ulimit` command for information on appropriate values.

Option	Description
`--cap-add=sys_nice`	Grants the container the `CAP_SYS_NICE` capability, which allows the container to raise process `nice` values, set real-time scheduling policies, set CPU affinity, and other operations.
`--cpu-rt-runtime=<value>`	The maximum number of microseconds the container can run at real-time priority within the Docker daemon's real-time scheduler period. You also need the `--cap-add=sys_nice` flag.
`--ulimit rtprio=<value>`	The maximum real-time priority allowed for the container. You also need the `--cap-add=sys_nice` flag.

The following example command sets each of these three flags on a `debian:jessie` container.

```
$ docker run -it \
    --cpu-rt-runtime=950000 \
    --ulimit rtprio=99 \
    --cap-add=sys_nice \
    debian:jessie
```

If the kernel or Docker daemon isn't configured correctly, an error occurs.

GPU

Access an NVIDIA GPU

Prerequisites

Visit the official NVIDIA drivers page to download and install the proper drivers. Reboot your system once you have done so.

Verify that your GPU is running and accessible.

Install nvidia-container-toolkit

Follow the official NVIDIA Container Toolkit installation instructions.

Expose GPUs for use

Include the `--gpus` flag when you start a container to access GPU resources. Specify how many GPUs to use. For example:

```
$ docker run -it --rm --gpus all ubuntu nvidia-smi
```

Exposes all available GPUs and returns a result akin to the following:

```
+-------------------------------------------------------------------------------
| NVIDIA-SMI 384.130                Driver Version: 384.130                  |
|-------------------------------+----------------------+----------------------
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|===============================+======================+======================
|   0  GRID K520           Off  | 00000000:00:03.0 Off |                  N/A
| N/A   36C  P0    39W / 125W |      0MiB /  4036MiB |      0%    Default |
+-------------------------------+----------------------+----------------------
+-------------------------------------------------------------------------------
| Processes:                                                       GPU Memory
|  GPU       PID   Type   Process name                             Usage    |
|===============================================================================
|  No running processes found
+-------------------------------------------------------------------------------
```

Use the `device` option to specify GPUs. For example:

```
$ docker run -it --rm --gpus device=GPU-3a23c669-1f69-c64e-cf85-44e9b07e7a2a ub
```

Exposes that specific GPU.

```
$ docker run -it --rm --gpus '"device=0,2"' ubuntu nvidia-smi
```

Exposes the first and third GPUs.

Note

NVIDIA GPUs can only be accessed by systems running a single engine.

Set NVIDIA capabilities

You can set capabilities manually. For example, on Ubuntu you can run the following:

```
$ docker run --gpus 'all,capabilities=utility' --rm ubuntu nvidia-smi
```

This enables the `utility` driver capability which adds the `nvidia-smi` tool to the container.

Capabilities as well as other configurations can be set in images via environment variables. More information on valid variables can be found in the nvidia-container-toolkit documentation. These variables can be set in a Dockerfile.

You can also use CUDA images which sets these variables automatically. See the official CUDA images NGC catalog page.

Theme: Dark