Runtime constraints on resources

The operator can also adjust the performance parameters of the container:

Option	Description
`-m`, `--memory=""`	Memory limit (format: `<number>[<unit>]`). Number is a positive integer. Unit can be one of `b`, `k`, `m`, or `g`. Minimum is 4M.
`--memory-swap=""`	Total memory limit (memory + swap, format: `<number>[<unit>]`). Number is a positive integer. Unit can be one of `b`, `k`, `m`, or `g`.
`--memory-reservation=""`	Memory soft limit (format: `<number>[<unit>]`). Number is a positive integer. Unit can be one of `b`, `k`, `m`, or `g`.
`--kernel-memory=""`	Kernel memory limit (format: `<number>[<unit>]`). Number is a positive integer. Unit can be one of `b`, `k`, `m`, or `g`. Minimum is 4M.
`-c`, `--cpu-shares=0`	CPU shares (relative weight)
`--cpu-period=0`	Limit the CPU CFS (Completely Fair Scheduler) period
`--cpuset-cpus=""`	CPUs in which to allow execution (0-3, 0,1)
`--cpuset-mems=""`	Memory nodes (MEMs) in which to allow execution (0-3, 0,1). Only effective on NUMA systems.
`--cpu-quota=0`	Limit the CPU CFS (Completely Fair Scheduler) quota
`--blkio-weight=0`	Block IO weight (relative weight) accepts a weight value between 10 and 1000.
`--blkio-weight-device=""`	Block IO weight (relative device weight, format: `DEVICE_NAME:WEIGHT`)
`--device-read-bps=""`	Limit read rate from a device (format: `<device-path>:<number>[<unit>]`). Number is a positive integer. Unit can be one of `kb`, `mb`, or `gb`.
`--device-write-bps=""`	Limit write rate to a device (format: `<device-path>:<number>[<unit>]`). Number is a positive integer. Unit can be one of `kb`, `mb`, or `gb`.
`--device-read-iops=""`	Limit read rate (IO per second) from a device (format: `<device-path>:<number>`). Number is a positive integer.
`--device-write-iops=""`	Limit write rate (IO per second) to a device (format: `<device-path>:<number>`). Number is a positive integer.
`--oom-kill-disable=false`	Whether to disable OOM Killer for the container or not.
`--oom-score-adj=0`	Tune container's OOM preferences (-1000 to 1000)
`--memory-swappiness=""`	Tune a container's memory swappiness behavior. Accepts an integer between 0 and 100.
`--shm-size=""`	Size of `/dev/shm`. The format is `<number><unit>`. `number` must be greater than `0`. Unit is optional and can be `b` (bytes), `k` (kilobytes), `m` (megabytes), or `g` (gigabytes). If you omit the unit, the system uses bytes. If you omit the size entirely, the system uses `64m`.

User memory constraints

We have four ways to set user memory usage:

Option	Result
memory=inf, memory-swap=inf (default)	There is no memory limit for the container. The container can use as much memory as needed.
memory=L<inf, memory-swap=inf	(specify memory and set memory-swap as `-1`) The container is not allowed to use more than L bytes of memory, but can use as much swap as is needed (if the host supports swap memory).
memory=L<inf, memory-swap=2*L	(specify memory without memory-swap) The container is not allowed to use more than L bytes of memory, swap *plus* memory usage is double of that.
memory=L<inf, memory-swap=S<inf, L<=S	(specify both memory and memory-swap) The container is not allowed to use more than L bytes of memory, swap *plus* memory usage is limited by S.

Examples:

```
$ docker run -it ubuntu:14.04 /bin/bash
```

We set nothing about memory, this means the processes in the container can use as much memory and swap memory as they need.

```
$ docker run -it -m 300M --memory-swap -1 ubuntu:14.04 /bin/bash
```

We set memory limit and disabled swap memory limit, this means the processes in the container can use 300M memory and as much swap memory as they need (if the host supports swap memory).

```
$ docker run -it -m 300M ubuntu:14.04 /bin/bash
```

We set memory limit only, this means the processes in the container can use 300M memory and 300M swap memory, by default, the total virtual memory size (--memory-swap) will be set as double of memory, in this case, memory + swap would be 2*300M, so processes can use 300M swap memory as well.

```
$ docker run -it -m 300M --memory-swap 1G ubuntu:14.04 /bin/bash
```

We set both memory and swap memory, so the processes in the container can use 300M memory and 700M swap memory.

Memory reservation is a kind of memory soft limit that allows for greater sharing of memory. Under normal circumstances, containers can use as much of the memory as needed and are constrained only by the hard limits set with the `-m`/`--memory` option. When memory reservation is set, Docker detects memory contention or low memory and forces containers to restrict their consumption to a reservation limit.

Always set the memory reservation value below the hard limit, otherwise the hard limit takes precedence. A reservation of 0 is the same as setting no reservation. By default (without reservation set), memory reservation is the same as the hard memory limit.

Memory reservation is a soft-limit feature and does not guarantee the limit won't be exceeded. Instead, the feature attempts to ensure that, when memory is heavily contended for, memory is allocated based on the reservation hints/setup.

The following example limits the memory (`-m`) to 500M and sets the memory reservation to 200M.

```
$ docker run -it -m 500M --memory-reservation 200M ubuntu:14.04 /bin/bash
```

Under this configuration, when the container consumes memory more than 200M and less than 500M, the next system memory reclaim attempts to shrink container memory below 200M.

The following example set memory reservation to 1G without a hard memory limit.

```
$ docker run -it --memory-reservation 1G ubuntu:14.04 /bin/bash
```

The container can use as much memory as it needs. The memory reservation setting ensures the container doesn't consume too much memory for long time, because every memory reclaim shrinks the container's consumption to the reservation.

By default, kernel kills processes in a container if an out-of-memory (OOM) error occurs. To change this behaviour, 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 is not set, this can result in the host running out of memory and require killing the host's system processes to free memory.

The following example limits the memory to 100M and disables the OOM killer for this container:

```
$ docker run -it -m 100M --oom-kill-disable ubuntu:14.04 /bin/bash
```

The following example, illustrates a dangerous way to use the flag:

```
$ docker run -it --oom-kill-disable ubuntu:14.04 /bin/bash
```

The container has unlimited memory which can cause the host to run out memory and require killing system processes to free memory. The `--oom-score-adj` parameter can be changed to select the priority of which containers will be killed when the system is out of memory, with negative scores making them less likely to be killed an positive more likely.

Kernel memory constraints

Kernel memory is fundamentally different than user memory as kernel memory can't be swapped out. The inability to swap makes it possible for the container to block system services by consuming too much kernel memory. Kernel memory includes :

- stack pages
- slab pages
- sockets memory pressure
- tcp memory pressure

1) When a user process needs to execute some privileged instruction (A system call) it traps to kernel mode and kernel executes the it on behalf of the user process. **This execution takes place on the processes' kernel stack.**

2) Frequently used objects in the Linux kernel have their own cache,

3) sockets memory pressure: some sockets protocols have memory pressure thresholds. The Memory Controller allows them to be controlled individually per cgroup, instead of globally.

4) tcp memory pressure: sockets memory pressure for the tcp protocol

You can setup kernel memory limit to constrain these kinds of memory. For example, every process consumes some stack pages. By limiting kernel memory, you can prevent new processes from being created when the kernel memory usage is too high.

Kernel memory is never completely independent of user memory. Instead, you limit kernel memory in the context of the user memory limit. Assume “U” is the user memory limit and “K” the kernel limit. There are three possible ways to set limits:

Option	Result
U != 0, K = inf (default)	This is the standard memory limitation mechanism already present before using kernel memory. Kernel memory is completely ignored.
U != 0, K < U	Kernel memory is a subset of the user memory. This setup is useful in deployments where the total amount of memory per-cgroup is overcommitted. Overcommitting kernel memory limits is definitely not recommended, since the box can still run out of non-reclaimable memory. In this case, you can configure K so that the sum of all groups is never greater than the total memory. Then, freely set U at the expense of the system's service quality.
U != 0, K > U	Since kernel memory charges are also fed to the user counter and reclamation is triggered for the container for both kinds of memory. This configuration gives the admin a unified view of memory. It is also useful for people who just want to track kernel memory usage.

Examples:

```
$ docker run -it -m 500M --kernel-memory 50M ubuntu:14.04 /bin/bash
```

We set memory and kernel memory, so the processes in the container can use 500M memory in total, in this 500M memory, it can be 50M kernel memory tops.

```
$ docker run -it --kernel-memory 50M ubuntu:14.04 /bin/bash
```

We set kernel memory without **-m**, so the processes in the container can use as much memory as they want, but they can only use 50M kernel memory.

Swappiness constraint

By default, a container's kernel can swap out a percentage of anonymous pages. To set this percentage for a container, specify a `--memory-swappiness` value between 0 and 100. A value of 0 turns off anonymous page swapping. A value of 100 sets all anonymous pages as swappable. By default, if you are not using `--memory-swappiness`, memory swappiness value will be inherited from the parent.

For example, you can set:

```
$ docker run -it --memory-swappiness=0 ubuntu:14.04 /bin/bash
```

Setting the `--memory-swappiness` option is helpful when you want to retain the container's working set and to avoid swapping performance penalties.

CPU share constraint

By default, all containers get the same proportion of CPU cycles. This proportion can be modified by changing the container's CPU share weighting relative to the weighting of all other running containers.

To modify the proportion from the default of 1024, use the `-c` or `--cpu-shares` flag to set the weighting to 2 or higher. If 0 is set, the system will ignore the value and use the default of 1024.

The proportion will only apply when CPU-intensive processes are running. When tasks in one container are idle, other containers can use the left-over CPU time. The actual amount of CPU time will vary depending on the number of containers running on the system.

For example, consider three containers, one has a cpu-share of 1024 and two others have a cpu-share setting of 512. When processes in all three containers attempt to use 100% of CPU, the first container would receive 50% of the total CPU time. If you add a fourth container with a cpu-share of 1024, the first container only gets 33% of the CPU. The remaining containers receive 16.5%, 16.5% and 33% of the CPU.

On a multi-core system, the shares of CPU time are distributed over all CPU cores. Even if a container is limited to less than 100% of CPU time, it can use 100% of each individual CPU core.

For example, consider a system with more than three cores. If you start one container `{C0}` with `-c=512` running one process, and another container `{C1}` with `-c=1024` running two processes, this can result in the following division of CPU shares:

```
PID    container    CPU CPU share
100    {C0}     0   100% of CPU0
```

```
101    {C1}     1   100% of CPU1
102    {C1}     2   100% of CPU2
```

CPU period constraint

The default CPU CFS (Completely Fair Scheduler) period is 100ms. We can use `--cpu-period` to set the period of CPUs to limit the container's CPU usage. And usually `--cpu-period` should work with –`cpu-quota`.

The **Completely Fair Scheduler** (**CFS**) is a process scheduler which was merged into the 2.6.23 release of the Linux kernel and is the default scheduler. It handles CPU resource allocation for executing processes,

```
Within each given "period" (microseconds), a group is allowed to consume only up
to "quota" microseconds of CPU time.  When the CPU bandwidth consumption of a
group exceeds this limit (for that period), the tasks belonging to its
hierarchy will be throttled and are not allowed to run again until the next
period.
```

Examples:

```
$ docker run -it --cpu-period=50000 --cpu-quota=25000 ubuntu:14.04 /bin/bash
```

If there is 1 CPU, this means the container can get 50% CPU worth of run-time every 50ms.

For more information, see the CFS documentation on bandwidth limiting.

```
cpu.cfs_quota_us: the total available run-time within a period (in microseconds)
cpu.cfs_period_us: the length of a period (in microseconds)

When the CPU bandwidth consumption of a group exceeds this limit [quota] (for
that period), the tasks belonging to its hierarchy will be throttled and are not
allowed to run again until the next period.
```

I'd say that they are trying to point out, that if you provision 10 seconds of CPU time in a minute, then the app can be stopped for 50 seconds as it is out of provisioned time. 10 seconds running uninterrupted at full power, 50 seconds nothing.

Cpuset constraint

We can set cpus in which to allow execution for containers.

Examples:

```
$ docker run -it --cpuset-cpus="1,3" ubuntu:14.04 /bin/bash
```

This means processes in container can be executed on cpu 1 and cpu 3.

```
$ docker run -it --cpuset-cpus="0-2" ubuntu:14.04 /bin/bash
```

This means processes in container can be executed on cpu 0, cpu 1 and cpu 2.

We can set mems in which to allow execution for containers. Only effective on NUMA systems.

Examples:

```
$ docker run -it --cpuset-mems="1,3" ubuntu:14.04 /bin/bash
```

This example restricts the processes in the container to only use memory from memory nodes 1 and 3.

```
$ docker run -it --cpuset-mems="0-2" ubuntu:14.04 /bin/bash
```

This example restricts the processes in the container to only use memory from memory nodes 0, 1 and 2.

CPU quota constraint

The `--cpu-quota` flag limits the container's CPU usage. The default 0 value allows the container to take 100% of a CPU resource (1 CPU). The CFS (Completely Fair Scheduler) handles resource allocation for executing processes and is default Linux Scheduler used by the kernel. Set this value to 50000 to limit the container to 50% of a CPU resource. For multiple CPUs, adjust the `--cpu-quota` as necessary. For more information, see the CFS documentation on bandwidth limiting.

Block IO bandwidth (Blkio) constraint

By default, all containers get the same proportion of block IO bandwidth (blkio). This proportion is 500. To modify this proportion, change the container's blkio weight relative to the weighting of all other running containers using the `--blkio-weight` flag.

> **Note:** The blkio weight setting is only available for direct IO. Buffered IO is not currently supported.

The `--blkio-weight` flag can set the weighting to a value between 10 to 1000. For example, the commands below create two containers with different blkio weight:

```
$ docker run -it --name c1 --blkio-weight 300 ubuntu:14.04 /bin/bash
$ docker run -it --name c2 --blkio-weight 600 ubuntu:14.04 /bin/bash
```

If you do block IO in the two containers at the same time, by, for example:

```
$ time dd if=/mnt/zerofile of=test.out bs=1M count=1024 oflag=direct
```

You'll find that the proportion of time is the same as the proportion of blkio weights of the two containers.

The `--blkio-weight-device="DEVICE_NAME:WEIGHT"` flag sets a specific device weight. The `DEVICE_NAME:WEIGHT` is a string containing a colon-separated device name and weight. For example, to set `/dev/sda` device weight to `200`:

```
$ docker run -it \
    --blkio-weight-device "/dev/sda:200" \
    ubuntu
```

If you specify both the `--blkio-weight` and `--blkio-weight-device`, Docker uses the `--blkio-weight` as the default weight and uses `--blkio-weight-device` to override this default with a new value on a specific device. The following example uses a default weight of `300` and overrides this default on `/dev/sda` setting that weight to `200`:

```
$ docker run -it \
    --blkio-weight 300 \
    --blkio-weight-device "/dev/sda:200" \
    ubuntu
```

The `--device-read-bps` flag limits the read rate (bytes per second) from a device. For example, this command creates a container and limits the read rate to `1mb` per second from `/dev/sda`:

```
$ docker run -it --device-read-bps /dev/sda:1mb ubuntu
```

The `--device-write-bps` flag limits the write rate (bytes per second)to a device. For example, this command creates a container and limits the write rate to `1mb` per second for `/dev/sda`:

```
$ docker run -it --device-write-bps /dev/sda:1mb ubuntu
```

Both flags take limits in the `<device-path>:<limit>[unit]` format. Both read and write rates must be a positive integer. You can specify the rate in `kb` (kilobytes), `mb` (megabytes), or `gb` (gigabytes).

The `--device-read-iops` flag limits read rate (IO per second) from a device. For example, this command creates a container and limits the read rate to `1000` IO per second from `/dev/sda`:

```
$ docker run -ti --device-read-iops /dev/sda:1000 ubuntu
```

The `--device-write-iops` flag limits write rate (IO per second) to a device. For example, this command creates a container and limits the write rate to `1000` IO per second to `/dev/sda`:

```
$ docker run -ti --device-write-iops /dev/sda:1000 ubuntu
```

Both flags take limits in the `<device-path>:<limit>` format. Both read and write rates must be a positive integer.

the operator can have fine grain control over the capabilities using `--cap-add` and `--cap-drop`. By default, Docker has a default list of capabilities that are kept. The following table lists the Linux capability options which can be added or dropped.

Capability Key	Capability Description
SETPCAP	Modify process capabilities.
SYS_MODULE	Load and unload kernel modules.
SYS_RAWIO	Perform I/O port operations (iopl(2) and ioperm(2)).
SYS_PACCT	Use acct(2), switch process accounting on or off.
SYS_ADMIN	Perform a range of system administration operations.

Capability Key	Capability Description
SYS_NICE	Raise process nice value (nice(2), setpriority(2)) and change the nice value for arbitrary processes.
SYS_RESOURCE	Override resource Limits.
SYS_TIME	Set system clock (settimeofday(2), stime(2), adjtimex(2)); set real-time (hardware) clock.
SYS_TTY_CONFIG	Use vhangup(2); employ various privileged ioctl(2) operations on virtual terminals.
MKNOD	Create special files using mknod(2).
AUDIT_WRITE	Write records to kernel auditing log.
AUDIT_CONTROL	Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.
MAC_OVERRIDE	Allow MAC configuration or state changes. Implemented for the Smack LSM.
MAC_ADMIN	Override Mandatory Access Control (MAC). Implemented for the Smack Linux Security Module (LSM).
NET_ADMIN	Perform various network-related operations.
SYSLOG	Perform privileged syslog(2) operations.
CHOWN	Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	Use RAW and PACKET sockets.
DAC_OVERRIDE	Bypass file read, write, and execute permission checks.
FOWNER	Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
DAC_READ_SEARCH	Bypass file read permission checks and directory read and execute permission checks.
FSETID	Don’t clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	Bypass permission checks for sending signals.
SETGID	Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	Make arbitrary manipulations of process UIDs.
LINUX_IMMUTABLE	Set the FS_APPEND_FL and FS_IMMUTABLE_FL i-node flags.
NET_BIND_SERVICE	Bind a socket to internet domain privileged ports (port numbers less than 1024).
NET_BROADCAST	Make socket broadcasts, and listen to multicasts.
IPC_LOCK	Lock memory (mlock(2), mlockall(2), mmap(2), shmctl(2)).
IPC_OWNER	Bypass permission checks for operations on System V IPC objects.
SYS_CHROOT	Use chroot(2), change root directory.
SYS_PTRACE	Trace arbitrary processes using ptrace(2).
SYS_BOOT	Use reboot(2) and kexec_load(2), reboot and load a new kernel for later execution.
LEASE	Establish leases on arbitrary files (see fcntl(2)).
SETFCAP	Set file capabilities.
WAKE_ALARM	Trigger something that will wake up the system.
BLOCK_SUSPEND	Employ features that can block system suspend.

Further reference information is available on the capabilities(7) - Linux man page

Both flags support the value `ALL`, so if the operator wants to have all capabilities but `MKNOD` they could use:

```
$ docker run --cap-add=ALL --cap-drop=MKNOD ...
```

For interacting with the network stack, instead of using `--privileged` they should use `--cap-add=NET_ADMIN` to modify the network interfaces.