How to Limit Resources in Docker?

written by sysadmin | 14 May 2025

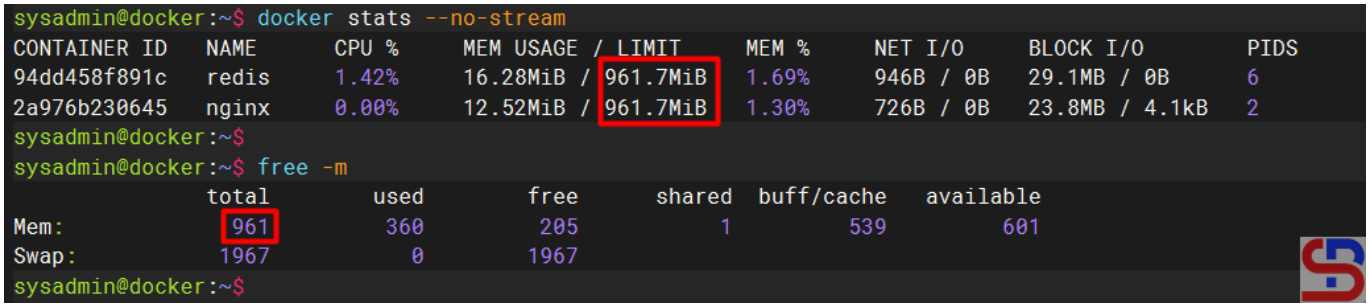
The previous article explained how to monitor an entire container in Docker. In addition to monitoring, you should also limit the resource usage of each container so that server performance remains in good condition.

Problem

How to limit resources in Docker?

Solution

If you run the docker stats command, the command will display the resource container as shown in the image below:



Display stats of the Docker

As you can see in the image above, there are 2 containers running, and both containers have a memory limit of 941 MB, where the memory size is the size of the memory of the server. This is dangerous because the application in the container can use all the memory or CPU on the server, causing the server to run abnormally. Therefore, you should limit the use of resources in the container.

A. RAM limitation

There are 2 types of memory limitations in Docker:

- The hard limit is a maximum value that cannot be

exceeded. When a container exceeds a hard memory limit, Docker takes aggressive actions such as terminating the container so that there will be an OOM or Out Of Memory error in the container. The options used in Docker are **--memory** or **-m**.

- The soft limit is a limit that can be temporarily exceeded. When a soft limit is reached, Docker warns the user but does not take immediate action. The option used is **--memory-reservation**.

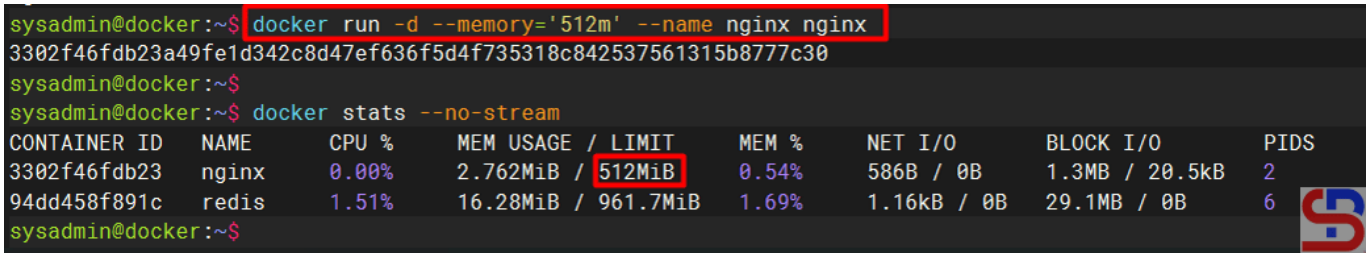
If you use swap on the server, you can use the **--memory-swap** option to allocate available swap memory to the container. This swap memory must be larger than the hard limit, usually twice larger than the hard limit. If you want to use a percentage for memory swap, use the **--memory-swappiness** option.

To limit the memory on the new container, use the format below:

```
docker run -d --memory='hard_limit_value' --name docker_name docker-image
```

For example, if you want to limit the memory on a container of 512 MB with a soft limit of 256 MB, then I run the command below:

```
docker run -d --memory='512m' --name nginx nginx
```



Run Docker with limited memory

You can immediately change the container memory when the container is running using the format below:

```
docker update docker_name --memory='hard_limit_value'
```

For example, I want to change the Redis container memory from 971 MB to 256 MB using the command below:

```
sysadmin@docker:~$ docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O        PIDS
3302f46fdb23   nginx     0.00%     2.762MiB / 512MiB     0.54%     936B / 0B     1.3MB / 20.5kB   2
94dd458f891c   redis     1.27%     16.28MiB / 961.7MiB   1.69%     1.23kB / 0B   29.1MB / 0B      6
sysadmin@docker:~$
sysadmin@docker:~$ docker update redis --memory='256m'
Error response from daemon: Cannot update container 94dd458f891c08d8e0a15eb00878fc83e5c66660d6af8beb0815a3a9040c57f7: Memo
ry limit should be smaller than already set memoryswap limit, update the memoryswap at the same time
sysadmin@docker:~$
```

Error when updating memory

But when running the command, there is an error as below:

Error Response from Daemon: Cannot Update Container 94DD458F891C08D8E0A15EB00878FC83E5C6660D6AF8Beb0815A3A9040C57F7: Memory Limit Should Be Smaller Than Already Set Time

To overcome the error, update the memory container swap, whose value must be greater than the memory value. Therefore, use the command below to increase the memory of a container:

```
docker update redis --memory='256m' --memory-swap='512mb'
```

And the container memory value should have changed as shown below:

```
sysadmin@docker:~$ docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O        PIDS
3302f46fdb23   nginx     0.00%     2.762MiB / 512MiB     0.54%     1.01kB / 0B   1.3MB / 20.5kB   2
94dd458f891c   redis     1.35%     16.28MiB / 961.7MiB   1.69%     1.3kB / 0B    29.1MB / 0B      6
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect redis | grep Memory
            "Memory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
sysadmin@docker:~$
sysadmin@docker:~$ docker update redis --memory='256m'
Error response from daemon: Cannot update container 94dd458f891c08d8e0a15eb00878fc83e5c66660d6af8beb0815a3a9040c57f7: Memo
ry limit should be smaller than already set memoryswap limit, update the memoryswap at the same time
sysadmin@docker:~$
sysadmin@docker:~$ docker update redis --memory='256m' --memory-swap='512mb'
redis
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect redis | grep Memory
            "Memory": 268435456,
            "MemoryReservation": 0,
            "MemorySwap": 536870912,
            "MemorySwappiness": null,
sysadmin@docker:~$
sysadmin@docker:~$ docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O        PIDS
3302f46fdb23   nginx     0.00%     2.762MiB / 512MiB     0.54%     1.01kB / 0B   1.3MB / 20.5kB   2
94dd458f891c   redis     1.56%     16.28MiB / 256MiB     6.36%     1.3kB / 0B    29.1MB / 0B      6
sysadmin@docker:~$
```

```
Update the memory in a container
```

B. CPU limitation

Before you limit the CPU in the container, you need to know how many CPU cores there are on your server by using the command below:

```
nproc
```

To limit the CPU used in the container, use the **--cpus** option to determine how many CPU cores can be used in a container. If your server has two CPUs and you set --cpus=’1.5′, the container is guaranteed at most one and a half of the CPUs and this is the equivalent of setting --cpu-period=’100000′ and --cpu-quota=’150000′ (cpu-period is used with cpu-quota to configure the CPU scheduler with defaults to 100000 microseconds and cpu-quota is used with cpu-period to configure the CPU scheduler). Use the format below to limit the CPU in the container:

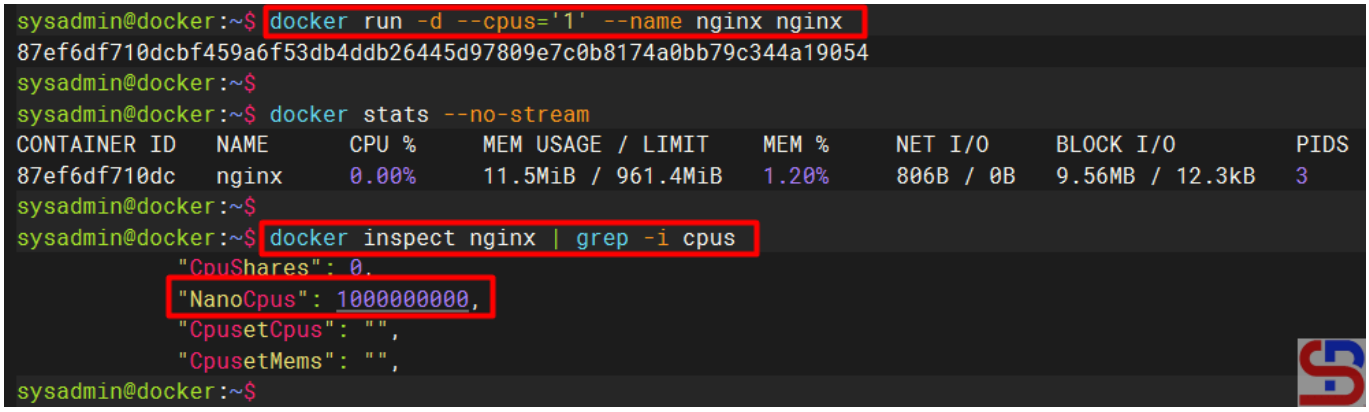
```
docker run -d --cpus='cpu_value' --name docker_name docker_image:tag
```

You can see the details of the CPU used by a container by using the format below:

```
docker inspect nginx | grep -e Cpu -e cpu
```

For example, I want to limit the CPU to 1, so I run the command below:

```
docker run -d --cpus='1' --name nginx nginx
```



Run a container with a limited CPU

Warning

You have to be careful in determining the CPU limitations of a container because if you limit the CPU too much (for example, giving 0.5 CPU to the container even though the container can run using 1 CPU) or give a CPU limitation that is greater than the CPU the server uses (for example The server CPU only has 1 CPU but you give the container 2 CPUs) then the container will immediately turn off automatically.

You can use the **--cpu-shares** option for a container to control the share of CPU cycles available, whose default value is 1024. This option is like a soft limit option on memory, so if you run the command below:

```
docker run -d --cpu-shares='2048' --name webapp1 nginx
```

If there is more than 1 container on a host and CPU cycles are constrained, then the webapp1 container will receive 2x more CPU than the other containers. You can update the CPU in a running container using the format:

```
docker update docker_name --cpus='cpu_value'
```

For example, I have 2 CPUs on the server, and initially, I use all the CPUs in the webapp1 container. Then, I wanted to update the CPU on the container to 1.5, so I ran the command below:

```
docker update webapp1 --cpus='1.5' nginx
```

```
sysadmin@docker:~$ docker run --name nginx -d nginx
8e506e9c6f058fbca872888e4f1db06c30836dc17b67a7a6e69f873eec476167
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect nginx | grep -i cpus
            "CpuShares": 0,
            "NanoCpus": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
sysadmin@docker:~$
sysadmin@docker:~$ docker update nginx --cpus="1.5"
nginx
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect nginx | grep -i cpus
            "CpuShares": 0,
            "NanoCpus": 1500000000,
            "CpusetCpus": "",
            "CpusetMems": "",
sysadmin@docker:~$
```

Update the CPU in a container

C. HDD limitation

As of this writing (April 2025), the HDD limitation can only be used for **btrfs**, **overlay2**, **windowsfilter**, and **zfs** storage drivers. For the overlay2 storage driver, the size option is only available if the backing filesystem is **xfs** and mounted with the **pquota** mount option. Type the command below to see the Docker settings on the server:

```
docker info | grep -e 'Filesystem' -e 'Support' -e 'Storage'
```

Run the command above, and here is the result:

```
sysadmin@docker:~$ docker info | grep -e 'Filesystem' -e 'Support' -e 'Storage'
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
sysadmin@docker:~$
```

Check the filesystem

To limit the hard disk in a container, follow the format below:

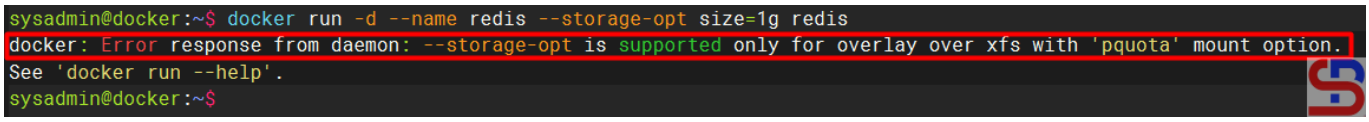
```
docker run -d --name nginx --storage-opt size=1g nginx
```

If I want to limit my hard disk in a container, I run the command below:

```
docker run -d --name nginx --storage-opt size=1g nginx
```

But, I have an error like the image below:

```
docker: Error response from daemon: –storage-opt is supported only for overlay pver xfs with ‘pquota’ mount option
```



Error when limiting HDD for a container

From the image above, I can’t limit the HDD to the container because my Backing Filesystem in my server still uses extfs, not xfs. So, if I want to limit my HDD in my containers, I have to change my Backing Filesystem in my server. So I made an experiment by turning off Docker and deleting the Docker folder using the command below:

```
sudo systemctl stop docker
sudo rm -rf /var/lib/docker && sudo mkdir /var/lib/docker
```

Then I inserted an additional hard disk into the existing server, and then I converted the file system to xfs using the commands below:

```
sudo mkfs.xfs /dev/sdb1
sudo mount /dev/sdb1 /var/lib/docker
```

```
sysadmin@docker:~$ sudo mount /dev/sdb1 /var/lib/docker/
sysadmin@docker:~$
sysadmin@docker:~$ df -T
Filesystem                         Type  1K-blocks    Used Available Use% Mounted on
tmpfs                              tmpfs     98448    1112     97336   2% /run
/dev/mapper/ubuntu--vg-ubuntu--lv ext4    10218772 6110044   3568056  64% /
tmpfs                              tmpfs    492220       0    492220   0% /dev/shm
tmpfs                              tmpfs      5120       0      5120   0% /run/lock
/dev/sda2                          ext4     1768056   96560   1563364   6% /boot
tmpfs                              tmpfs     98444      12     98432   1% /run/user/1000
/dev/sdb1                          xfs     2030592   71980   1958612   4% /var/lib/docker
sysadmin@docker:~$
```

Create xfs filesystem

And I added to the **/etc/fstab** file the script below:

```
echo '/dev/sdb1 /var/lib/docker xfs defaults,quota,prjquota,pquota,gquota 0
0' | sudo tee -a /etc/fstab
```

I rebooted the server to test whether the fstab file settings were correct. After that, I tried to create a container by limiting the container's hard disk to 1GB using the command below:

```
docker run -d --name nginx --storage-opt size=1g nginx
```

```
sysadmin@docker:~$ docker info | grep -e 'Filesystem' -e 'Support' -e 'Storage'
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
sysadmin@docker:~$
sysadmin@docker:~$ docker run -d --name nginx --storage-opt size=1g nginx
8d609d92bcc7d2b6be5d0c3f888ad6f7d6135f05466ded4761cb6a6941c59a17
sysadmin@docker:~$
sysadmin@docker:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
8d609d92bcc7   nginx     "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    80/tcp    nginx
sysadmin@docker:~$
```

Limiting HDD in a container

Note

You can combine more than one restriction above by using one command. For example, if you want to limit memory, CPU, and HDD in a container, then you can run the command below:

```
docker run -d --name webapp1 -m 512m --cpus=1.5 --storage-opt size=1g nginx
```

As far as I know, Sysadmin only limits the use of RAM and CPU in Docker, but rarely limits HDD in Docker. If you want to limit resources in Docker, you must discuss it with the developers so that there are no problems with the application in the future.

References

phoenixnap.com
baeldung.com
docs.docker.com
nodramadevops.com
stackoverflow.com
hands-on.cloud
blogs.perficient.com
blog.devops.dev
youtube.com
reddit.com