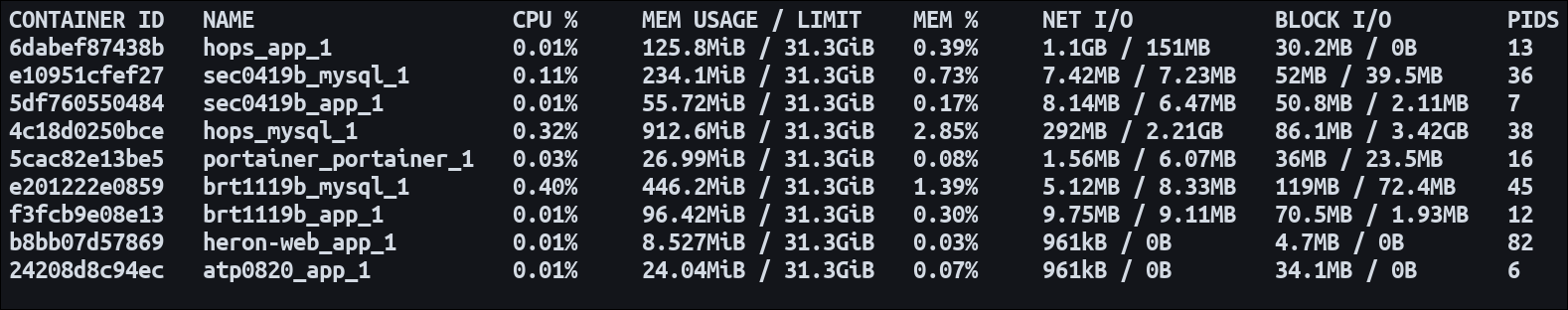
**Docker Monitoring**

**The Docker Stats Command:**

Docker’s built-in mechanism for viewing resource consumption is docker stats. This command gives you a tabulated view of your containers. Each container displays a live feed of its critical metrics.

The command’s output includes CPU consumption and a measure of each container’s network and storage use during its lifetime. The Memory column shows the live memory usage as well as the memory limit configured on the container. When no limit is set, you’ll see the amount of RAM available on your host. The final column, PIDS, is a count of the number of processes started by the container.



Stopped containers are excluded by default. You can add them to the table by passing the -a (--all) flag to the command. CPU and memory use will be unavailable, but you’ll be able to see the metrics that are aggregated through the container’s life, such as network activity.

You can view the stats of single and multiple containers in the same way as other common docker CLI commands. Pass a list of space-separated container IDs or names. The output will show the metrics for the specified containers, removing everything else.

|  |
| --- |
| **$ docker stats first-container second-container**  **$ docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"**  **$ docker stats --format "{{.Container}}: {{.CPUPerc}}"**  **$ docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"**  **On Linux:**  "table{{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}\t{{.NetIO}}\t{{.BlockIO}}\t{{.PIDs}}"  **On Windows:**  "table{{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}\t{{.BlockIO}}" |

Docker exposes metrics via three mechanisms: pseudo-files in sysfs, the stats command, and API. Metrics coverage across these three mechanisms is uneven, as seen below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access via | CPU metrics | Memory metrics | I/O metrics | Network metrics |
| pseudo-files | Yes | Yes | Some | Yes, as of v1.6.1 |
| stats command | Basic | Basic | Some, as of v1.9.0 | Basic |
| API | Yes | Yes | Some | Yes |

**Pseudo-files:**

Docker metrics reported via pseudo-files in sysfs by default do not require privileged (root) access. They are also the fastest and most lightweight way to read metrics; if you are monitoring many containers per host, speed may become a requirement. However, you cannot collect all metrics from pseudo-files. As seen in the table above, there may be limitations on I/O and network metrics.

**Pseudo-file location:**

This article assumes your metrics pseudo-files are located in /sys/fs/cgroup in the host OS. In some systems, they may be in /cgroup instead.

Your pseudo-file access path includes the long id of your container. For illustration purposes this article assumes that you have set an env variable CONTAINER\_ID to the long ID of the container you are monitoring. If you’d like to copy-paste run commands in this article, you can set CONTAINER\_ID like this: CONTAINER\_ID=$(docker run [OPTIONS] IMAGE [COMMAND] [ARG...] )

or

you can save it after launching:

docker ps --no-trunc and then copy-paste and save the long ID as an env variable like CONTAINER\_ID=<long ID>

**CPU pseudo-files:**

CPU metrics are reported in cpu and cpuacct (CPU accumulated). In the commands below, we use the metric directory for standard Linux systems (/sys/fs/cgroup/cpuacct/docker/$CONTAINER\_ID/).

Usage:

|  |
| --- |
| **$ cat /sys/fs/cgroup/cpuacct/docker/$CONTAINER\_ID/cpuacct.stat**  > user 2451 # time spent running processes since boot  > system 966 # time spent executing system calls since boot |

**CPU Usage per core:**

Per-CPU usage can help you identify core imbalances, which can be caused by bad configuration.

|  |
| --- |
| **$ cat /sys/fs/cgroup/cpuacct/docker/$CONTAINER\_ID/cpuacct.usage\_percpu**  > 45094018900 # nanoseconds CPU has been in use since boot (45.09s) |

If your container is using multiple CPU cores and you want a convenient total usage number, you can run:

|  |
| --- |
| $ cat /sys/fs/cgroup/cpuacct/docker/$CONTAINER\_ID/cpuacct.usage  > 45094018900 # total nanoseconds CPUs have been in use (45.09s |

**Throttled CPU:**

If you set a limit on the CPU time available to a container with CPU quota constraint, your container will be throttled when it attempts to exceed the limit.

|  |
| --- |
| $ cat /sys/fs/cgroup/cpu/docker/$CONTAINER\_ID/cpu.stat  > nr\_periods 565 # Number of enforcement intervals that have elapsed  > nr\_throttled 559 # Number of times the group has been throttled  > throttled\_time 12119585961 # Total time that members of the group were throttled, in nanoseconds (12.12 seconds) |

**Memory pseudo-files:**

The following command will print a lot of information of memory usage, probably more than you need. Note that the first half of the measures have no standard prefix; these measures exclude sub-cgroups. The second half all are prefixed with “total\_”; these measures include sub-cgroups.

|  |
| --- |
| $ cat /sys/fs/cgroup/memory/docker/$CONTAINER\_ID/memory.stat  cache 532480  rss 10649600  mapped\_file 1576960  writeback 0  swap 0  pgpgin 302242  pgpgout 296556  pgfault 1142200  pgmajfault 125  inactive\_anon 16384  active\_anon 577536  inactive\_file 11386880  active\_file 11309056  unevictable 0  hierarchical\_memory\_limit 18446744073709551615  hierarchical\_memsw\_limit 18446744073709551615  total\_cache 22798336  total\_rss 491520  total\_rss\_huge 0  total\_mapped\_file 1576960  total\_writeback 0  total\_swap 0  total\_pgpgin 302242  total\_pgpgout 296556  total\_pgfault 1142200  total\_pgmajfault 125  total\_inactive\_anon 16384  total\_active\_anon 577536  total\_inactive\_file 11386880  total\_active\_file 11309056  total\_unevictable 0 |

You can get most interesting memory metrics directly by calling a specific command in the /sys/fs/cgroup/memory/docker/$CONTAINER\_ID/ directory:

|  |
| --- |
| # Total memory used: cached + rss  $ cat /sys/fs/cgroup/memory/docker/$CONTAINER\_ID/memory.usage\_in\_bytes    # Total memory used + swap in use  $ cat /sys/fs/cgroup/memory/docker/$CONTAINER\_ID/memory.memsw.usage\_in\_bytes    # Number of times memory usage hit limts  $ cat /sys/fs/cgroup/memory/docker/$CONTAINER\_ID/memory.failcnt    # Memory limit of the cgroup in bytes  $ cat /sys/fs/cgroup/memory/docker/$CONTAINER\_ID/memory.limit\_in\_bytes |

Note that if the final command returns a long garbage number like 18446744073709551615, you did not set the limit when you launched the container. To set a 500MB limit, for example:

|  |
| --- |
| $ docker run -m 500M IMAGE [COMMAND] [ARG...] |

[**I/O pseudo-files**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#io-pseudo-files)**:**

The path to I/O stats pseudo-files for most operating systems is:

/sys/fs/cgroup/blkio/docker/$CONTAINER\_ID/.

Depending on your system, you may have many metrics available from these pseudo-files:

blkio.io\_queued\_recursive, blkio.io\_service\_time\_recursive, blkio.io\_wait\_time\_recursive and more. On many systems, however, many of these pseudo-files only return zero values. In this case there are usually still two pseudo-files that work: blkio.throttle.io\_service\_bytes and blkio.throttle.io\_serviced, which report total I/O bytes and operations, respectively. Contrary to their names, these numbers do not report throttled I/O but actual I/O bytes and ops.

The first two numbers reported by these pseudo-files are the major: minor device IDs, which uniquely identify a device. Example output from blkio.throttle.io\_service\_bytes:

253:0 Read 13750272

253:0 Write 180224

253:0 Sync 180224

253:0 Async 13750272

253:0 Total 13930496

[**Network pseudo-files**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#network-pseudo-files)**:**

[Docker version 1.6.1 and greater](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#docker-version-161-and-greater), In [release 1.6.1](https://github.com/docker/docker/blob/master/CHANGELOG.md#161-2015-05-07), Docker fixed read/write /proc paths.

|  |
| --- |
| $ CONTAINER\_PID= `docker inspect -f '{{ .State.Pid }}' $CONTAINER\_ID`  $ cat /proc/$CONTAINER\_PID/net/dev |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inter-| | Receive | | | | | | | | Transmit | | | | | | | |
| face| | bytes | packets | errs | drop | fifo | frame | compressed | multicast| | bytes | packets | errs | drop | fifo | colls | carrier | compressed |
| eth0: | 1296 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 816 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| lo: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[**Older versions of Docker**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#older-versions-of-docker)**:**

You can get network metrics from ip netns, with some symlinking:

|  |
| --- |
| $ CONTAINER\_PID=`docker inspect -f '{{ .State.Pid }}' $CONTAINER\_ID`  $ mkdir -p /var/run/netns  $ ln -sf /proc/$CONTAINER\_PID/ns/net /var/run/netns/$CONTAINER\_ID  $ ip netns exec $CONTAINER\_ID netstat -i |

[**Stats command**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#stats-command)**:**

The docker stats command will continuously report a live stream of basic CPU, memory, and network metrics. As of [version 1.9.0](https://github.com/docker/docker/releases/tag/v1.9.0), docker stats also includes disk I/O metrics.

# Usage: docker stats CONTAINER [CONTAINER...]

|  |
| --- |
| $ **docker stats $CONTAINER\_ID**  CONTAINER CPU % MEM USAGE/LIMIT MEM % NET I/O BLOCK I/O  ecb37227ac84 0.12% 71.53 MiB/490 MiB 14.60% 900.2 MB/275.5 MB 266.8 MB/872.7 MB |

[**CPU stats**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#cpu-stats)**:**

CPU is reported as % of total host capacity. So if you have two containers each using as much CPU as they can, each allocated the same CPU shares by docker, then the stat command for each would register 50% utilization, though in practice their CPU resources would be fully utilized.

[**Memory stats**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#memory-stats)**:**

If you do not explicitly set the memory limits for the container, then the memory usage limit will be the memory limit of the host machine. If the host is using memory for other processes, your container will run out of memory before it hits the limit reported by the stats command.

[**I/O stats**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#io-stats)**:**

As of Docker version 1.9.0, docker stats now displays total bytes read and written.

[**Network stats**](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#network-stats)**:**

Displays total bytes received (RX) and transmitted (TX).

[Requirements](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/#requirements): Docker version 1.5.0 (released February 2015) or higher

============END============