

# Docker Stats | Understand how to monitor Docker Metrics with docker stats

Link: <https://signoz.io/blog/docker-stats/>

Docker containers are transient (lasting for a very short time), spawning quickly and in high numbers, which causes metrics bursts. This makes monitoring a challenge due to Docker's scaling and redeployment features. Docker stats is a built-in feature of Docker containers. The `docker stats` command returns a live data stream of your running containers.

Docker is a containerization platform that lets you separate your applications from your infrastructure to deliver software quickly. To monitor Docker, it is crucial to gather performance-related metrics from various system elements, such as containers, hosts, and databases.

## Methods of monitoring Docker Metrics

There are several ways in which Docker metrics can be monitored. These include;

- Docker stats command
- Pseudo-files in sysfs
- REST API exposed by the Docker daemon

Note that these are in-built features of Docker.

### “ Observability for your containerized application

Observability is critical for modern cloud-native applications. It helps engineering teams have more confidence in their production environment. Troubleshooting performance issues is easier with a robust observability framework in place.

SigNoz, an open-source observability tool, can help make your containerized applications observable.

To get started with SigNoz, please visit the [documentation](#).

In this article, we will deep dive into the `docker stats` command that can be used to monitor Docker metrics right from the terminal.

# What is the `docker stats` command?

The `docker stats` command is a built-in feature of Docker that displays resource consumption statistics for the container in real-time. By default, it shows CPU and memory utilization for all containers. Stats here refers to “statistics”. You can restrict the statistics by entering the container names or IDs you're interested in monitoring. The `docker stats` command output includes CPU stats, memory metrics, block I/O, and network IO metrics for all active containers.

## A Practical Approach

Running the `docker stats` command produces an output that looks like the code snippet below. This command shows the stats for `all` the running Docker containers.

```
$ docker stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM%	NET I/O	BLOCK
56b3f523b0sd	nginx-container	0.35%	2.534MiB / 16.455GiB	0.37%	568B / 0B	
134kb / 0B	3					
049996113bc8	ubuntu	0.14%	1.437MiB / 16.455GiB	0.10%	3.56kb / 0B	
5.12MB / 0B	1					
a3f78cb32a8e	hello-world	0.00%	1.228MiB / 16.455GiB	0.06%	65.45kb / 0B	550kb
/ 0B	0					

# Understanding the docker stats command output

The `docker stats` command returns a live snapshot of resource usage by Docker containers. Let's break down all the stats given by the command.

## CPU% stats

CPU is expressed as a percentage (%) of the overall host capacity. One can optimize the resource usage of Docker hosts by being aware of how much CPU the hosts and containers consume. One active/busy container shouldn't slow down other containers by consuming all of the CPU resources. Containers can be optimized based on the amount of CPU they are using.

## MEM USAGE / LIMIT Stats

`MEM USAGE` lists the available memory. It gives a quick overview of the container's memory usage and allocation, providing information about the container's memory statistics, including usage and memory limit. Except when it is defined for a specific container, the memory usage limit corresponds to the host machine's memory limit.

## MEM % Stat

`MEM %` shows the memory percentage that the container is using from its host machine.

## Network(NET) I/O Stats

`NET I/O` shows the volume of information the container's network interface has transmitted(TX) and received (RX). It represents network traffic.

## BLOCK I/O Stats

**BLOCK I/O** helps to identify containers that are writing data and shows the total number of bytes read and written to the container file system. Block I/O stats can give you an idea about issues with data persistence.

# PIDS

PIDS is a count of the processes that the container has created or the number of kernel process IDs running inside the corresponding container.

## More on the usage of `docker stats`

## Getting stats of a particular container

To get the stats of a particular container, provide the container Id and run the command `docker stats <containerID>`

```
$ docker stats 56b3f523b0sd
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM%	NET I/O	BLOCK
I/O	PIDS					
56b3f523b0sd	nginx-container	0.35%	2.534MiB / 16.455GiB	0.37%	568B / 0B	
134kb / 0B	3					

□ □

You can also get the stats of multiple containers by name and id if you run

```
docker stats <containerName> <containerId>
```

```
$ docker stats ubuntu 56b3f523b0sd
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM%	NET I/O	BLOCK
I/O	PIDS					
049996113bc8	ubuntu	0.14%	1.437MiB / 16.455GiB	0.10%	3.56kb / 0B	
5.12MB / 0B	1					
56b3f523b0sd	nginx-container	0.35%	2.534MiB / 16.455GiB	0.37%	568B / 0B	
134kb / 0B	3					

# Display options that Docker Provides

These display options allow you to specify how you want the output to be shown.

Docker stats offers the following options for display:

1. `--all` which shows all containers, whether stopped or running.
2. `--format` which uses the [Go Template syntax](#) to print images out.
3. `--no-stream` which disables streaming stats and only pulls the first result
4. `--no-trunc` which instructs Docker not to truncate (shorten) output.

The syntax for this is shown below:

```
$ docker stats [OPTIONS] [CONTAINER...]
```

■

## Using `docker stats --format`

Let's take a look at the `--format` option.

Docker format is used to modify the output format of commands that have the `--format` option. If a command supports this option, it can be used to change the output format of the command to suit our needs since the default command does not display all the fields connected to that object.

By using the Go Template syntax, the formatting option `--format` presents container output in an easy-to-read way.

For example,

```
$ docker stats --format "{{.Container}}: {{.CPUPerc}}"
```

```
049996113bc8: 0.14%
```

```
56b3f523b0sd: 0.35%
```

■

This prints out all images with the Container and CPUPerc (CPU Percentage) elements, separated by a colon (:) and it uses a template without headers.

To display all container information in a table format, including name, CPU percentage, and memory consumption, use the following syntax:

```
$ docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"
```

CONTAINER ID	CPU %	PRIV WORKING SET
56b3f523b0sd	0.35%	2.534MiB / 16.455GiB
049996113bc8	0.14%	1.437MiB / 16.455GiB
a3f78cb32a8e	0.00%	1.228MiB / 16.455GiB



Here is the list of applicable placeholders to use with the Go template syntax:

Placeholder	Description
.container	Container name or ID (user input)
.Name	Container name
.ID	Container ID
.CPUPerc	CPU percentage
.MemUsage	Memory usage
.NetIO	Network IO
.BlockIO	Block IO
.MemPerc	Memory percentage (Not available on Windows)
.PIDs	Number of PIDs (Not available on Windows)

## Final Thoughts

In this article, we discussed ways to monitor resource usage metrics in Docker focused on the `docker stats` command. Other ways of using The Docker stats, Pseudo-files in sysfs, and REST API exposed by the Docker daemon are native ways of monitoring resource utilization metrics.

Docker container monitoring is critical for running containerized applications. For a robust monitoring and observability setup, you need to use a tool that visualizes the metrics important for container monitoring and also lets you set alerts on critical metrics. SigNoz is an open-source observability tool that can help you do that.

It uses [OpenTelemetry](#) to collect metrics from your containers for monitoring. OpenTelemetry is becoming the world standard for instrumentation of cloud-native applications, and it is backed by [CNCF](#) foundation, the same foundation under which Kubernetes graduated.

If you want to set up a robust observability framework for your containerized application, you can use SigNoz. You can create unified views to monitor your Docker containers effectively.

It is easy to get started with SigNoz. It can be installed on macOS or Linux computers in just three steps by using a simple installation script.

The install script automatically installs Docker Engine on Linux. However, you must manually install Docker Engine on macOS before running the install script.

```
git clone -b main https://github.com/SigNoz/signoz.git
cd signoz/deploy/
./install.sh
```



---

Revision #1

Created 5 May 2024 11:01:03 by Administrador

Updated 4 July 2024 18:59:31 by Administrador