How to Run HAProxy With Docker (In-Depth Guide)

Link: https://www.haproxy.com/blog/how-to-run-haproxy-with-docker

August 1st, 2025
8 min read

Nick Ramirez

Can you run HAProxy as a Docker container? Yes! Did you even need to ask? Docker is ubiquitous these days and you'll find that many applications have been *Docker-ized*; the HAProxy load balancer is no exception. Pardon the cliché, but HAProxy was born for this. As a standalone service that runs on Linux, porting it to Docker certainly seemed natural.

Why would you want to run your load balancer inside of a Docker container? Are their performance penalties when doing so? Will it introduce any security issues?

In this blog post, you'll learn why you might consider running HAProxy inside a container and what the ramifications could be. Then you'll see how to do it. Note that if you want to use HAProxy for Kubernetes traffic management, we encourage you to view our Kubernetes solution to see what's possible.

HAProxy Technologies builds its own set of Docker images under its namespace *haproxytech*. These are updated regularly with the latest patches and security updates. I will be using those images in this blog post. You'll find them here:

- HAProxy (Alpine Linux base)- https://hub.docker.com/r/haproxytech/haproxy-alpine
- HAProxy (Ubuntu base) – https://hub.docker.com/r/haproxytech/haproxy-ubuntu
- HAProxy (Debian base) – https://hub.docker.com/r/haproxytech/haproxy-debian

The commands I demonstrate were performed on a Linux workstation but will work just as well when using Docker Desktop for Windows or Docker Desktop for Mac.

?#The Benefits of Docker

Do you want the ability to run HAProxy without needing to compile it, install dependencies, or otherwise alter your system?

Docker containers bring considerable benefits, chief among them being less ceremony around installation and execution. Docker allows you to drop a container onto a host system and instantly get a running service—no install scripts, no installing C libraries. The service is completely contained within the container and all you need to do is start it and then map a TCP port to it. When you deploy a container, you gain the ability to run an entire application complete with its runtime environment without ever actually installing it onto the host system.

Lifecycle management becomes standardized too. Starting, stopping, and removing a container is as easy as calling one-line *docker* commands. That in turn makes deployment a repeatable and testable process. It also lends itself to easier software upgrades.

?#The Performance Impact of Running Docker

You want your load balancer to be fast, with no added latency from the environment. So, the question is, what is the impact of running HAProxy inside of a container?

In terms of CPU overhead, it helps to remember that, unlike a virtual machine, Docker does not require a layer of virtualization on top of the host operating system. A container runs on the host's kernel and is basically just another process, albeit one with better isolation from other processes running on the host (it uses *namespaces* to accomplish this). It should come as little surprise then that a study by researchers at IBM found that the CPU overhead of using Docker is negligible.

Networking is another story. By default, Docker lets you access the services running inside containers by creating a bridge network to the host. This does incur latency due to the network address translation (NAT) that must happen between the container's local network and the host's bridge network. In the same IBM study cited before, the researchers found that Docker's NAT doubled latency from roughly 35 µs to 70 µs for a 100-byte request from the client and a 200-byte response from the application.

On the other hand, bridge networks are useful because they allow you to isolate groups of containers into a container-only network and expose only some of those containers to the host, which is handy for reducing the number of IP addresses required on your host's network (think about the number of IPs required to run hundreds or possibly thousands of containers).

If you require very low latency you can switch to using Docker's host network feature, which allows your container to share the same network as the host, cutting out the need for NAT. Then again, that doesn't touch on what to do if you want to run Docker Swarm or Kubernetes, which use overlay networks, for which different network drivers like Project Calico and Cilium have solutions. However, that is outside the scope of this article.

In short, unless you require very low latency, you should be fine sticking with the default bridge networking option. Just be sure to test it out and see if you're getting the throughput you need.

?#Security Considerations of Running Docker

You may be concerned by the fact that many Docker containers run their service as *root*, and this root user is the same root user as on the host system. Concerns about a container breakout are legitimate. HAProxy runs as root too. However, to put your mind at ease: HAProxy requires root access because it needs to bind to restricted TCP ports like 80 and 443. However, once it has finished its startup, it drops its root privileges and runs as an unprivileged user.

People also weigh the risk that a container may be malicious. This is a good reason to stick with the *haproxytech* Docker images, which are curated by HAProxy Technologies.

?#Run Haproxy With Docker

We'll create three instances of a web application, one instance of HAProxy, and a bridge network to join them together. So, once you've installed Docker, use the following command to create a new bridge network in Docker:

```
$ sudo docker network create --driver=bridge mynetwork
```

view raw

Then use the `docker run` command to create and run three instances of the web application. In this example, I use the Docker image jmalloc/echo-server. It's a simple web app that returns back the details of the HTTP requests that you send to it.

```
$ sudo docker run -d \
--name web1 --net mynetwork jmalloc/echo-server:latest

$ sudo docker run -d \
--name web2 --net mynetwork jmalloc/echo-server:latest

$ sudo docker run -d \
--name web3 --net mynetwork jmalloc/echo-server:latest
```

view raw

Notice that we assign each one a unique name and attach it to the bridge network we created. You should now have three web applications running, which you can verify by calling the `docker ps` command:

```
$ sudo docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
98216bb8c5ff jmalloc/echo-server:latest "/bin/echo-server" About a minute ago Up About a minute 8080/tcp web3
ae6accc111d9 jmalloc/echo-server:latest "/bin/echo-server" About a minute ago Up About a minute 8080/tcp web2
554fafbc2b3b jmalloc/echo-server:latest "/bin/echo-server" About a minute ago Up About a minute 8080/tcp web1
```

view raw

These containers listen on their own port 8080, but we did not map those ports to the host, so they are not routable. We'll relay traffic to these containers via the HAProxy load balancer. Next, let's add HAProxy in front of them. Create a file named haproxy.cfg in the current directory and add the following to it:

```
global
stats socket /var/run/api.sock user haproxy group haproxy mode 660 level admin expose-fd listeners
log stdout format raw local0 info

defaults
mode http
timeout client 10s
timeout connect 5s
timeout server 10s
timeout http-request 10s
log global

frontend stats
bind *:8404
stats enable
stats uri /
stats refresh 10s

frontend myfrontend
bind :80
default_backend webservers

backend webservers
server s1 web1:8080 check
server s2 web2:8080 check
server s3 web3:8080 check
```

view raw

A few things to note:

- In the `global` section, the `stats socket` line enables the HAProxy Runtime API and also enables seamless reloads of HAProxy.
- The first frontend listens on port 8404 and enables the HAProxy Stats dashboard, which displays live statistics about your load balancer.

- The other frontend listens on port 80 and dispatches requests to one of the three web applications listed in the *webservers* backend.
- Instead of using the IP address of each web app, we're using their hostnames *web1*, *web2* , and *web3*. You can use this type of DNS-based routing when you create a Docker bridge network as we've done.

Next, create and run an HAProxy container and map its port 80 to the same port on the host by including the `-p` argument. Also, map port 8404 for the HAProxy Stats page:

```
$ sudo docker run -d \
--name haproxy \
--net mynetwork \
-v $(pwd):/usr/local/etc/haproxy:ro \
-p 80:80 \
-p 8404:8404 \
haproxytech/haproxy-alpine:2.4
```

view raw

Calling `docker ps` afterwards shows that HAProxy is running:

```
$ sudo docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
d734d0ef2635 haproxytech/haproxy-alpine:2.4 "/docker-entrypoint.…." 3 seconds ago Up 2 seconds 0.0.0.0:80->80/tcp, 0.0.0.0:8404->8404/tcp haproxy
```

view raw

You can access the *echo-server* web application at http://localhost. Each request to it will be load balanced by HAProxy. Also, you can see the HAProxy Stats page at http://localhost:8404.

If you make a change to your haproxy.cfg file, you can reload the load balancer—without disrupting traffic—by calling the `docker kill` command:

```
$ sudo docker kill -s HUP haproxy
```

view raw

To delete the containers and network, run the `docker stop`, `docker rm`, and `docker network rm` commands:

```
$ sudo docker stop web1 && sudo docker rm web1
$ sudo docker stop web2 && sudo docker rm web2
$ sudo docker stop web3 && sudo docker rm web3
$ sudo docker stop haproxy && sudo docker rm haproxy
$ sudo docker network rm mynetwork
```

view raw

?#Conclusion

In this blog post, you learned how running HAProxy inside of a Docker container can simplify its deployment and lifecycle management. Docker provides a standardized way for deploying applications, making the process repeatable and testable. While the CPU overhead of running Docker is negligible, it can incur extra network latency, but the impact of that depends on your use case and throughput needs.

To run HAProxy, simply create an HAProxy configuration file and then call the `docker run` command with the name of the HAProxy Docker image. HAProxy Technologies supplies up-to-date Docker images on Docker Hub.

Want to know when more content like this is published? Subscribe to our blog or follow us on Twitter. You can also join the conversation on Slack.

Get the latest release updates, tutorials, and deep-dives from HAProxy experts.

Authors Nick Ramirez

Nick creates technical content for HAProxy Technologies ranging from documentation and blog posts to Wikipedia articles, GitHub READMEs and Stack Overflow answers. With a background in web development and DevOps, he has fun digging into product features and discovering the optimal path for a new blog tutorial.

Twitter LinkedIn GitHub

Related Posts

October 3rd, 2024

Use the GeoIP Database With HAProxy (Easy to Follow Guide)

Geolocation is the process of linking a third party to a geographical location. In easier words: know the country of a client's IP address. On the Internet, such a base is called GeoIP.

December 13th, 2023

Hitless Reloads With HAProxy (How-To Configuration Guide)

HAProxy offers a patch set for enabling seamless reloads of HAProxy without dropping packets in the process. In this blog post, we show you how to enable this.

September 10th, 2024

HAProxy on Docker Swarm: load balancing & DNS service discovery

In this blog post, you'll see how to combine HAProxy and Docker Swarm to load balance traffic across your service replicas.

August 1st, 2025

How to Check & Test Your HAProxy Config Safely (Easy Guide)

Learn how to safely test and validate your HAProxy config file with a single command.

Revision #2
Created 26 August 2025 16:20:05 by Administrador
Updated 26 August 2025 16:27:25 by Administrador